JIISIC'07

VI Jornadas Iberoamericanas de Ingeniería del Software e Ingeniería del Conocimiento

Lima-Perú
31 de enero al 2 de febrero de 2007

Editado y Compilado por:

Facultad de Ciencias e Ingeniería
Departamento de Ingeniería
Maynard Kong
José Antonio Pow-Sang
Manuel Francisco Tupia
Luis Alberto Flores

VI Jornadas Iberoamericanas de Ingeniería del Software e Ingeniería del Conocimiento-JIISIC'07

Compilado por:
Facultad de Ciencias e Ingeniería de la Pontificia Universidad Católica del Perú
Departamento de Ingeniería de la Pontificia Universidad Católica del Perú

Editado por:
Facultad de Ciencias e Ingeniería de la Pontificia Universidad Católica del Perú
Departamento de Ingeniería de la Pontificia Universidad Católica del Perú
Maynard Kong Wong, José Antonio Pow-Sang Portillo, Manuel Francisco Tupia Anticona y Luis Alberto Flores García.

Primera edición: enero de 2007

Hecho el Depósito Legal en la Biblioteca Nacional del Perú Nº2007-00571

ISBN N° 978-9972-2885-1-7

Comité Permanente

Silvia Teresita Acuña, Universidad Autónoma de Madrid, España
Manoel Mendonça, Universidade Salvador, Brasil
Oscar Dieste, Universidad Complutense de Madrid, España

Comité Organizador

José Antonio Pow-Sang, Pontificia Universidad Católica del Perú **(chair)**
Manuel Tupia, Pontificia Universidad Católica del Perú
Luis Flores, Pontificia Universidad Católica del Perú
Felipe Solari, Pontificia Universidad Católica del Perú

Comité de Programa

Maynard Kong, Pontificia Universidad Católica del Perú, Perú **(chair)**
Raul Aguilar, Universidad Autonoma de Yucatán, México
Idoia Alarcon, Universidad Autónoma de Madrid, España
Luis Alberto Alvarez, Universidad Austral, Chile
Marco Alvarez, Utah State University, EEUU
Pedro Antunes, Universidade de Lisboa, Portugal
Joao Araujo, Universidade Nova de Lisboa, Portugal
Marianela Aveledo, Universidad Simon Bolivar, Venezuela
Pere Botella, Universitat Politècnica de Catalunya, España
David Camacho, Universidad Autonoma de Madrid, España
Francisco Camargo, ITESM, México
Zalatiel Carranza, Universidad de Lima, Perú
Dante Carrizo, Universidad Complutense de Madrid, España
Luca Cernuzzi, Univ. Católica Ntra. Señora de la Asunción, Paraguay
Sergio Coronado, University of Luxembourg, Luxemburgo
Ernesto Cuadros-Vargas, Universidad Católica San Pablo, Perú
Angelica de Antonio, Universidad Politécnica de Madrid, España
Amador Duran, Universidad de Sevilla, España
Juan Vicente Echagüe, Universidad de la República, Uruguay
Yadran Eterovic, Pontificia Universidad Catolica de Chile, Chile
Mariano Fernandez, Universidad CEU San Pablo, España
Xavier Ferre, Universidad Politécnica de Madrid, España
Ramon Garcia, Instituto Tecnológico de Buenos Aires, Argentina
Francisco Jose Garcia, Universidad de Salamanca, España
Luis Guerrero, Universidad de Chile, Chile
Ricardo Imbert, Universidad Politécnica de Madrid, España
Mario Jino, Universidade Estadual de Campinas, Brasil
Nora La Serna, Universidad Nacional Mayor de San Marcos, Perú
Guillermo Licea, Universidad Autónoma de Baja California, México
Marta Lopez, Universidad Complutense de Madrid, España
Jose Antonio Macias, Universidad Autónoma de Madrid, España
Esperanza Marcos, Universidad Rey Juan Carlos, España
Victor Hugo Medina, Universidad Distrital Fco. José Caldas, Colombia
Nelson Medinilla, Universidad Politécnica de Madrid, España
Ana María Moreno, Universidad Politécnica de Madrid, España
Jaime Muñoz, Universidad Autónoma de Aguascalientes, México
Melvin Perez, CAM Informatica, República Dominicana
Claudia Pons, Universidad Nacional de la Plata, Argentina
Angel Puerta, Redwhale Software , EEUU
Isidro Ramos, Universitat Politècnica de Valencia, España
Gustavo Rodríguez, INAOE, México
Maria Isabel Sanchez Segura, Universidad Carlos III de Madrid, España

Comité de Programa (continuación)

René Santaolaya Salgado, CENIDET, México
Miguel Angel Serrano, CIMAT, México
Almudena Sierra, Universidad Rey Juan Carlos, España
Enrique Sierra, Instituto Tecnológico de Buenos Aires, Argentina
Francisco Tirado, Universidad Complutense de Madrid, España
Ambrosio Toval, Universidad de Murcia, España
Jorge Triñanes, Universidad de la República, Uruguay
Raimundo Vega, Universidad Austral, Chile
Sira Vegas, Universidad Politécnica de Madrid, España
Silvia Regina Vergilio, Universidade Federal do Paraná, Brasil
Monica Villavicencio, Escuela Superior Politécnica del Litoral, Ecuador
Marcello Visconti, Universidad Técnica Federico Santa Maria, Chile
Aurora Vizcaíno Barceló, Universidad de Castilla-La Mancha, España

Colaboradores en el Proceso de Revisión

Abel Gómez
Alejandro Hossian
Alex Bustos
Aurora Pozo
César J. Acuña
Enrique Fernandez
Fernando Molina
Fuensanta Medina Domínguez
Jaime Navón
Jennifer Pérez
Joaquín Nicolás
José Ángel Olivas
Jose Arturo Mora Soto
Jose Carsí
José María Cavero
Luis Flores
Manuel Tupia
Maria Alejandra Ochoa
Marisa Cogliati
Miguel Ángel Martínez Aguilar
Nelly Condori-Fernandez
Norberto Millo
Paola Britos
Percy Pari Salas
Sonia Pamplona

Prólogo

Este volumen contiene los trabajos aceptados y presentados en las VI Jornadas Iberoamericanas de Ingeniería del Software e Ingeniería del Conocimiento (JIISIC'07) celebradas en Lima, Perú, del 31 de enero al 2 de febrero de 2007. Desde su edición inicial en 2001, las JIISIC han demostrado ser el foro de reunión más importante, a nivel Iberoamericano, de investigadores y profesionales interesados en ambas disciplinas.

El evento actual es la continuación de la labor iniciada en las JIISIC'01, celebrada en en Buenos Aires (Argentina), JIISIC'02 en Salvador de Bahía (Brasil), JIISIC'03 en Valdivia (Chile), JIISIC'04 en Madrid (España) y JIISIC'06 en Puebla (México).

En la presente convocatoria se han recibido 88 artículos de calidad científica para su evaluación. Cada trabajo ha sido evaluado por al menos 2 revisores y se ha contemplado la resolución de divergencias, que por cierto han sido muy pocas. Finalmente fueron aceptados 54 artículos de autores procedentes de Argentina, Brasil, Colombia, Corea del Sur, Cuba, Chile, Ecuador, España, Estados Unidos de América, Mexico, Perú y Uruguay. Además de la sesiones técnicas, se aceptaron cuatro tutoriales.

Es preciso indicar que todo esto no hubiera sido posible sin la colaboración de muchas personas. Por ello queremos agradecer especialmente a los miembros del Comité de Programa por su excelente y desinteresada labor, necesaria para renovar la calidad y prestigio ganado. También queremos destacar el enorme esfuerzo de Manuel Tupia, Luis Flores y Felipe Solari, miembros del Comité Organizador, sin cuyo trabajo no hubieran podido celebrarse estas Jornadas. Nuestro agradecimiento al Ing. Eduardo Ismodes, decano de la Facultad de Ciencias e Ingeniería, y al Ing. Kurt Paulsen, jefe del Departamento de Ingeniería, por el gran apoyo que nos han brindado. Por último, pero no al final, expresamos nuestro sincero agradecimiento a todos los autores que aportaron sus contribuciones al evento.

Maynard Kong
Presidente del Comité de Programa

José Antonio Pow-Sang
Presidente del Comité Organizador

ÍNDICE

Sesión 2b: Ontologías, Metodologías, Patrones y Frameworks

Sesión 2c: Ingeniería de Requerimientos, Arquitecturas y Diseño de Software

Sesión 3a: Arquitecturas y Diseño de Software

Sesión 3b: Métodos de Diseño, Modelado de Dominio y Meta-Modelado

Sesión 3c: Calidad en el Software

Sesión 4a: Modelado de Procesos

Sesión 4b: Ing. del Software basada en Componentes, Usabilidad e Interacción Persona-Computadora

Sesión 4c: Métricas e Ingeniería del Software Empírica

Sesión 5a: Modelos de Calidad

Sesión 5b: Métricas e Ingeniería del Software Empírica

Sesion 5c: Modelado y Mejora de Procesos

Sesión 6a: Modelos de Calidad

Sesión 6b: Aplicaciones Industriales y Cómputo Móvil

Sesión 6c: Educación en Ing. de Software e Ing. del Conocimiento, Informática Educativa

Sesión 6d: Mejora de Procesos

Tutoriales

ARTÍCULOS

Un Modelo de Proceso para Educción de Requisitos en Proyectos de Data Mining

José Gallardo Arancibia
Ing. de Sistemas y Comp.
U. Católica del Norte, Av. Angamos 0610,
Antofagasta, Chile
jgallardo@ucn.cl

Óscar Marbán Gallego
Facultad de Informática U. Politécnica de Madrid,
Campus de Montegancedo s/n, Madrid, España
omarban@fi.upm.es

Claudio Meneses Villegas
Ing. de Sistemas y Comp.
U. Católica del Norte, Av. Angamos 0610,
Antofagasta, Chile
cmeneses@ucn.cl

Resumen

El proceso de educción de requisitos, es una actividad compleja en cualquier tipo de proyectos y además, una de las más importantes por su connotación económica. En el caso de los Sistemas de Data Mining (DM), la complejidad se magnifica pues, con frecuencia ni el mismo cliente tiene claro lo que quiere. Lo mencionado refuerza la importancia de contar con un modelo de proceso que permita cumplir con esta importante actividad, de una manera sistemática y organizada. En este artículo se propone y describe un Modelo de Proceso para la Educción de Requisitos en Proyectos de Data Mining. Este modelo, se apoya en las actividades fundamentales de un proceso de Ingeniería de Requisitos (IR). El trabajo descrito, forma parte de un proyecto de investigación orientado a la definición de una Metodología para la construcción del Documento de Requisitos en Proyectos de Data Mining.

1. Introducción

Cuando se toma la decisión de construir un edificio, un sistema de software, un vehículo o cualquier producto en general, descubrir los requisitos que deberá cumplir el nuevo producto antes de iniciar su construcción, es una cuestión de suma importancia.

Sin embargo, esta afirmación que parece ser muy lógica y obvia, y que constituye una práctica habitual en diversas áreas tales como la construcción (no se construye un edificio sin un plano), la industria automotriz [18], comercio electrónico [5], o data warehouse [13] entre otras, es un aspecto que frecuentemente se soslaya en el desarrollo de Sistemas de Software, peor aun, en sistemas de Data Mining (DM) no existe un procedimiento o metodología ad-hoc, que permita identificar, capturar, verificar y validar los correctos requisitos de una manera sistemática.

A pesar de que la educción de requisitos es una actividad explicita en cualquier modelo de proceso de software, no necesariamente es considerada con la frecuencia y formalidad que amerita. Debbie Richards en [12] plantea que si bien, múltiples mejoras se han realizado en diversas actividades involucradas en el proceso de desarrollo de software, la captura, análisis y modelado de los requisitos de usuario, son las actividades menos exploradas aun.

Pero, ¿qué consecuencias ocasiona la omisión o una inadecuada definición de requisitos?. Según un estudio del Instituto Nacional de Estándares y Tecnología del Departamento de Comercio de los Estados Unidos [11], los errores y fallas imprevistas cuestan a la economía nacional unos 59.500 millones de dólares al año. La investigación también permitió constatar, que más de la mitad de todos los errores, no se encuentran hasta que el proceso de desarrollo está en su fase final o durante el período de uso, luego de la comercialización del software. En [14] se afirma, *"el descubrir los requisitos durante la construcción de un producto o pero aun, cuando el cliente empieza a utilizarlo es muy caro e ineficiente"*.

¿Qué ocurre en el área de Data Mining? En la última década, una gran cantidad de proyectos de Data Mining han sido desarrollados y se espera que en el futuro cercano esta cantidad aumente hasta en un 300%, así lo estima un informe de GartnerGroup [4]. Sin embargo, la ejecución de este tipo de proyectos, ha debido enfrentar una serie de problemas, no todos los proyectos se concluyen, o lo hacen fuera de todo plazo y con presupuestos no previstos [19]. Como una manera de enfrentar estos problemas, un grupo de empresas europeas pioneras en este tipo de proyectos

(Teradata, SPSS, Daimler-Chrysler y OHRA), propuso en 1999 una guía de referencia denominada CRISP-DM (Cross-Industry Standard Process for Data Mining) [2]. Esta guía presenta una recomendación destinada a permitir el desarrollo sistemático de este tipo de proyectos.

A grandes rasgos, se puede establecer que CRISP-DM, se presenta en términos de un proceso jerárquico, consistente de un conjunto de acciones descritas en diferentes niveles de abstracción, de lo más general a lo más específico.

CRISP-DM no es la única guía que ha sido propuesta. También existen otras propietarias o abiertas, como la desarrollada por la empresa SAS, denominada SEMMA (Sample, Explore, Modify, Model, Assess) [16], DMAMC [6] o las 5 A's [8]. Todas estas metodologías sin embargo, adolecen de métodos o técnicas que permitan educir adecuadamente los requisitos del proyecto. Más concretamente, aún no existe un proceso maduro que pueda calificarse como una metodología sólida, pues si bien por ejemplo, CRISP-DM, establece un conjunto de tareas y actividades que deben ser ejecutadas en el proyecto, no establece con qué técnicas o modelos se debe implementar cada actividad.

En el presente artículo se realiza una revisión de los procesos de Ingeniería de Requisitos (IR) más utilizados, se plantean los aspectos previos que deben considerarse antes de proceder a capturar los requisitos y luego se propone un Modelo de Proceso ad - hoc, para la Educción de Requisitos en Proyectos de Data Mining.

2. Modelos del proceso de Ingeniería de Requisitos (IR)

La Ingeniería de Requisitos, es una técnica actualmente muy utilizada por muchos especialistas para la construcción del Documento de Requisitos, el cual debe constituir el punto de partida para el correcto diseño e implementación de un sistema, cualquiera sea su naturaleza.

En el proceso de Ingeniería de Requisitos, se pueden identificar ciertos elementos o actividades fundamentales que deben ser desarrolladas para construir un documento de especificación de requisitos. Estas actividades fundamentales, como son la elicitación, el análisis, la especificación y la validación de requisitos, sirven de base a la propuesta de diversos modelos.

En [5], se plantea en base a las actividades de elicitación, especificación y validación, el modelo de proceso representado en la figura 1.

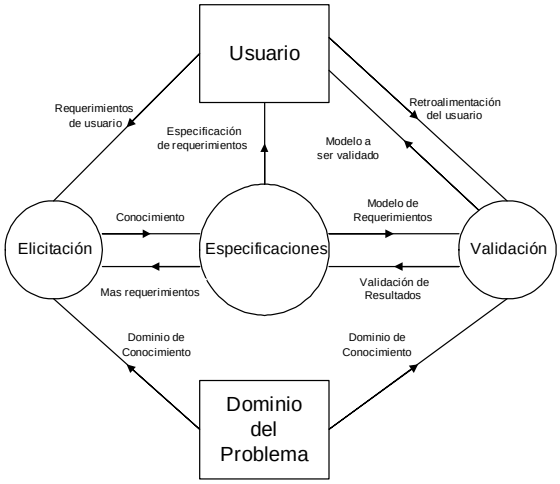


Figura 1. Esquema general del proceso de Ingeniería de Requisitos

Los elementos fundamentales del diagrama planteado, se describen brevemente como sigue [1]:

- *Elicitación:* Es el proceso de adquirir todo el conocimiento relevante, necesario para producir un modelo de los requisitos en un dominio del problema, mediante la comunicación con clientes, usuarios del sistema y quienes estén involucrados con el proyecto. Luego de obtenido un conjunto inicial de requisitos, estos deben ser *analizados* y representados en un lenguaje más técnico a fin de evitar inconsistencias y ambigüedades con el objeto de negociar un acuerdo.
- *Especificación:* El proceso del elicitación, proporciona la entrada para el proceso de especificación de requisitos. La salida, es un modelo de la especificación o los modelos que corresponden a diversas visiones. Estos modelos formalizan el conocimiento tácito del grupo o partes involucradas en el proyecto. La especificación de requisitos además, tiene un doble propósito, por un lado sirve como un acuerdo para el problema a ser resuelto entre el grupo involucrado en el proyecto y por otro, sirve como modelo para continuar con el siguiente paso.
- *Validación:* La validación, es la actividad destinada a comprobar si la especificación de requisitos, esta de acuerdo a lo esperado por los clientes. Además revisa que no se haya omitido ningún requisito y que éstos no sean ambiguos, inconsistentes o redundantes. En esta etapa se produce la integración y validación final de lo obtenido en las etapas anteriores, entregando

como resultado final, el Documento de Requisitos.

Ian Sommerville, [17] plantea otro modelo para el proceso de Ingeniería de Requisitos, tal como el representado en la figura 2.

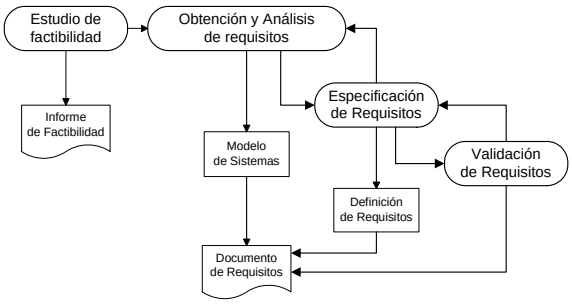


Figura 2. Proceso de Ingeniería de Requisitos

En este modelo, se incorpora en forma preliminar como primera etapa del proceso, un estudio de factibilidad. El estudio de factibilidad, es un primer estudio que recibe como entrada, una descripción breve del sistema a desarrollar y cómo éste será utilizado por la organización mandante. Su objetivo, es abordar aspectos relacionados con la factibilidad técnica y presupuestaria para el desarrollo del proyecto, bajo la consideración de que éste debe contribuir a los objetivos organizacionales, y la forma en que éste podría ser integrado a los sistemas ya existentes.

Cuando ya se cuenta con la información requerida, se procede a elaborar el informe de factibilidad. Este informe, debe incluir recomendaciones de cuando continuar con el desarrollo del proyecto, cambios en el alcance, presupuesto, calendarización del desarrollo del sistema y requisitos adicionales de alto nivel.

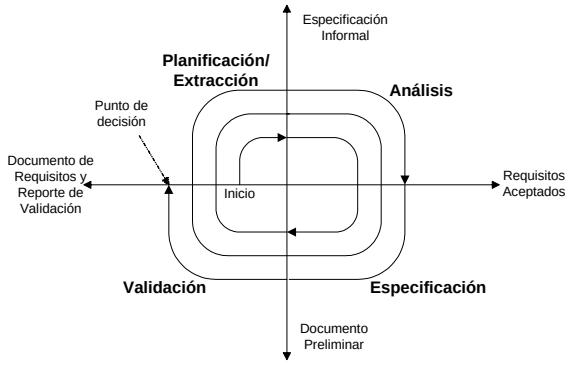


Figura 3. Modelo en Espiral del proceso de IR

Un tercer modelo que se puede analizar, es el modelo denominado "modelo en espiral" [7], representado en la figura 3. El uso de una espiral en este modelo, sirve para representar el que las diferentes actividades que componen el modelo, son actividades repetitivas, hasta el momento en que se toma la decisión final de aceptación del documento de especificación de requisitos. En este sentido, es importante destacar que el proceso puede ser influenciado por ciertos factores externos, que determinarían una finalización anticipada del proceso.

Finalmente, otro de los modelos de proceso que es importante destacar, es el proceso VOLERE y su plantilla asociada. El método VOLERE, desarrollado por James y Suzanne Robertson [14], consiste en un marco de trabajo para la adquisición y el análisis de requisitos de un sistema. Sus principales subsistemas son el Shell y la Plantilla de especificación de requisitos.

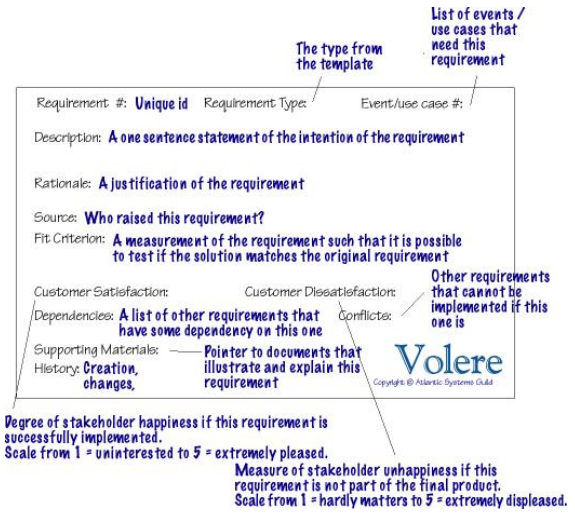


Figura 4. Descripción gráfica del Shell de Volere [14].

El Shell de Volere, consta de nueve componentes principales los cuales son: *los requisitos, la descripción, los fundamentos, las fuentes, criterios apropiados, clientes (satisfacción y descontento), las dependencias, materiales de soporte y la historia.*

La Plantilla de VOLERE, también se compone de un conjunto de componentes fundamentales, estos son: *las restricciones del producto, los requisitos funcionales y no funcionales, conductores del proyecto y tópicos del proyecto*.

La figura 4, describe gráficamente los componentes que constituyen el Shell de Volere.

Resumiendo, se puede establecer que no existe un modelo único que se pueda aplicar a todos los procesos de gestión de requisitos, por cuanto la realidad de las diversas organizaciones es diferente, diferentes sus objetivos y el tipo de proyectos o sistemas a ser desarrollados (operacionales o no operacionales).

3. Consideraciones preliminares para el planteamiento del modelo de IR para DM

El establecimiento de los requisitos en un proyecto de Data Mining, constituye una de las tareas de mayor trascendencia. La realización de esta tarea, involucra el especificar y validar los servicios que deberá proporcionar el sistema, como también identificar las restricciones que deberán considerarse en su desarrollo. Este proceso es esencial, debido a que los errores más comunes y costosos de corregir, son producto de una inadecuada Ingeniería de Requisitos.

Para poder educir satisfactoriamente los requisitos de un proyecto de DM, es importante que los responsables de este proceso, tengan un completo conocimiento de los objetivos de negocio y del dominio del problema. Normalmente, los administradores y los expertos de negocio, quienes son los más capacitados para definir la misión y visión de una organización y los encargados de generar las estrategias para el futuro de la organización, debieran poder transmitir de una manera precisa, lo que ellos esperan de un proyecto de DM a los desarrolladores del proyecto, sin embargo, si bien ellos entienden intuitivamente como trabaja su negocio, en muchos casos, no tienen la capacidad para comunicar de una manera clara y precisa esta información.

Por otro lado los expertos en DM, deberían poder vislumbrar de qué manera un Sistema de DM podría contribuir al cumplimiento de las metas de la organización, pero este es un esfuerzo, que en muchas ocasiones no se logra. La construcción de un modelo de negocios (figura 5) por lo tanto, servirá como un puente que permita una mejor y mutua comunicación entre ambas comunidades.

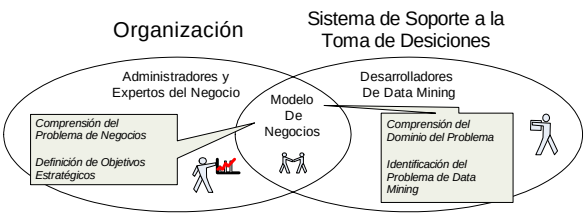


Figura 5. Nexo entre los expertos de negocio y desarrolladores de Data Mining.

Luego de construido el modelo de negocios de la organización, los expertos de negocio y los desarrolladores de Data Mining, podrán contar con un instrumento común que les facilite la definición de los objetivos estratégicos de la organización y una mejor comprensión de la forma en que un proyecto de Data Mining, pueda contribuir al logro de los objetivos señalados.

Las actividades relevantes para el proyecto y que permite potenciar un modelo de negocios son entonces:

i. *Identificar los objetivos del negocio.* Un proyecto de Data Mining, debe tener como objetivo final, generar algún tipo de beneficio para la organización, ya sea mejorando la eficiencia de los procesos del negocio o descubriendo nuevas fuentes de beneficio.

ii. *Identificar el dominio del problema.* En este contexto, la identificación del dominio del problema, permitirá especificar el área en que el proyecto de Data Mining tendrá lugar. A modo de ejemplo, una clasificación bastante general de dominios de problemas que un proyecto de Data Mining permite enfrentar es la siguiente.

1. *Marketing.* Considera la obtención de la mayor cantidad de información relacionada con los clientes del negocio para establecer potenciales clientes, determinar quien comprará, cuando y donde, mejorar la relación con los clientes, etc., es decir, el resultado del Data Mining, deberá permitir planificar de la mejor forma, las futuras campañas de marketing de la compañía.
2. *Predicción.* Considera la determinación a priori, del comportamiento futuro de una determinada variable de interés, que puede tener una fuerte relevancia para, por ejemplo, prevenir problemas, detectar oportunidades de negocio, optimizar inventarios, etc.
3. *Reducción de riesgos*. El Data Mining, permitirá una evaluación automática de riesgos, en base a experiencias previas.
4. *Detección de fraudes.* Podrán obtenerse modelos que permitan descubrir posibles fraudes, en base a modelos de detección de comportamientos anómalos.
5. *Control de calidad.* Considera la definición de modelos que permitirán, la detección precisa y anticipada de productos defectuosos.

iii. *Mapear el problema de negocios, en un problema de Data Mining*. Luego de identificado el dominio del problema, éste debe ser mapeado en un

problema de Data Mining, es decir, se debe considerar que cada tipo de problema, es soportado por algún tipo de enfoque algorítmico. Algunos enfoques son:

1. *Asociación.* Este enfoque algorítmico, es utilizado básicamente para descubrir relaciones entre atributos. Es decir, la idea es descubrir reglas que identifiquen patrones de comportamiento.
2. *Secuenciación.* Es similar al de asociación, sin embargo en este enfoque se incorpora la variable tiempo.
3. *Clasificación.* Este enfoque, emplea el conjunto de datos para desarrollar un modelo y utilizarlo como clasificador para nuevos conjuntos de datos.
4. *Regresión.* Un enfoque de regresión, se utiliza en los casos en que la salida predictiva puede tomar posibles valores ilimitados es decir, se trabaja fundamentalmente con variables continuas.
6. *Agrupamiento.* Este enfoque es utilizado en aplicaciones típicas de segmentación, y consiste en una partición de los datos en colecciones de datos relacionados o grupos en los cuales, los datos comparten un número de características comunes.

iv. *Definir las componentes que deberá considerar un requisito de Data Mining.* tales como:

1. *Componente de datos*. Esta componente debe responder a la pregunta sobre qué datos y con qué estructura (modelo de datos) se precisan, en función del enfoque algorítmico que soportará la aplicación de Data Mining.
2. *Componente de interfaz.* Debe responder a la pregunta sobre la forma en que serán visualizados o presentados los resultados del proyecto.
3. *Componente de usabilidad y correctitud.* Considera la forma en que el proyecto, debe responder las consultas que contribuyan al objetivo de negocio, y el grado de exactitud que proveerá el modelo.
4. *Componente de comprensibilidad.* Considera la manera en la cual, el modelo de Data Mining, permitirá justificar los resultados logrados.
5. *Componente de recursos.* Debe considerar los recursos disponibles para el proyecto, tanto en personal (expertos de negocio, de datos, de Data Mining, y asistencia técnica), como de plataformas de hardware y software.
6. *Componentes no funcionales*. Son requisitos de reutilización, entornos de desarrollo, disponibilidad y calidad de los resultados (plazos de entrega), seguridad y legislación.

4. Modelo Propuesto

La definición de requisitos es un proceso poco estructurado y de naturaleza iterativa, razones que justifican plantear un esquema en el cual, el proceso se estructure en una secuencia de fases, donde cada fase deba considerar determinados elementos de entrada, la realización de tareas y actividades bien definidas, la obtención de subproductos intermedios y la salida final del proceso.

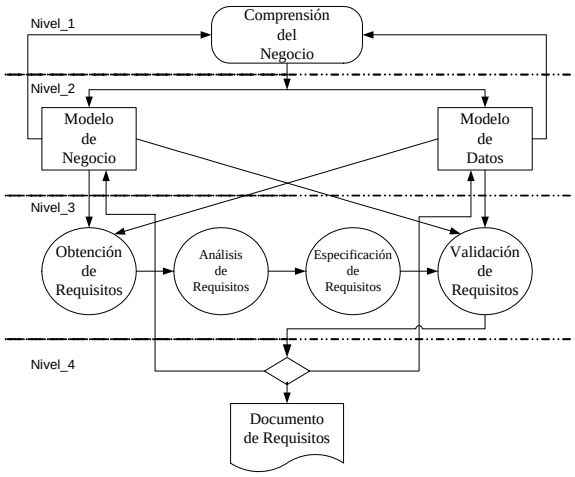


Figura 6. Modelo de Proceso de IR en Data Mining

Un modelo de proceso que recoge las consideraciones discutidas en el punto 3, se plantea en la figura 6. Este modelo se estructura en cuatro niveles, en los cuales se ejecutan una o un conjunto de tareas o fases para producir finalmente el documento de requisitos. En el primer nivel se define la fase con la cual parte el proceso y esta representada por la fase de comprensión del negocio. Esta fase, es una de las más importantes, pues de su pleno conocimiento podrán derivarse en el segundo nivel, los *Modelos de Negocio Decisional y de Datos,* que sirven de entrada a las fases definidas en el tercer nivel y que son las actividades típicas definidas en un modelo de Ingeniería de Requisitos. Las fases contempladas en este tercer nivel son la de Educción, Análisis, Especificación y Validación de los requisitos. Una vez que los requisitos se han validado, se concluye en el último nivel, en el que se encuentra la última fase del

proceso, que corresponde a la construcción del documento final.

Una descripción de cada una de las fases que componen el modelo es la siguiente:

Comprensión del negocio: Uno de los principales problemas cuando se inicia el proceso de educción de requisitos, lo constituye la mala comunicación que se establece, entre quien realiza el levantamiento de los requisitos y los clientes y usuarios, debido a ignorancias mutuas (el ingeniero de requisitos tiene un dominio de conocimiento diferente al del cliente o usuario), no existe un lenguaje común que permita una comunicación fluida. Por esta razón, en esta fase, quienes estén a cargo del levantamiento de requisitos, deberán interiorizarse del dominio del negocio (con incluso alguna corta estadía en la organización), de su estructura, procedimientos, políticas de mediano y largo plazo y cultura organizacional, para generar los necesarios lazos de confianza con directivos y quienes potencialmente se vean involucrados en el proyecto. Es en esta fase, donde también deben determinarse el o los problemas que deben ser resueltos y si es posible transformarlos en un problema de Data Mining.

Modelo de Negocios: Un Modelo de Negocio, puede tener una estructura particular, en función de un determinado aspecto o situación que se quiera destacar en el negocio, (procesos, reglas, objetivos, etc.). Considerando que Data Mining, es esencialmente una técnica que apoya el proceso de toma de decisiones en la organización, es necesario definir un Modelo de Negocios Decisional, en el cual este representado el negocio desde la perspectiva de la toma de decisiones. Este modelo, deberá estar compuesto por diferentes vistas, en las cuales estén identificadas las diferentes funciones y niveles dentro de la organización, y que requieran de sistemas de apoyo a la toma de decisiones [8], esto es, niveles *ejecutivo* (estratégico), *gerencial* (de administración), o *de conocimientos* (trabajadores de conocimiento y datos). A partir de este modelo y del modelo de datos, deberán posteriormente definirse los requisitos y restricciones a ser considerados en el modelo de Data Mining.

Modelo de Datos: Un sistema de Data Mining, es un sistema de tipo Decisional, por tanto su función principal, se circunscribe esencialmente a la comprensión y exploración de los datos de manera de determinar, de qué forma la información y conocimiento oculto en ellos, impactan en la toma de decisiones. En esta fase, se debe realizar un estudio que permita determinar, cuáles de los datos con que cuenta la organización, son los requeridos para el logro de los objetivos del proyecto, (los datos deben contener información que sea apropiada, consistente, limpia y representativa para la aplicación) generando posteriormente un esquema de representación adecuado en función del modelo con el cual se trabajará en el proyecto.

Las fases de *Obtención de Requisitos*, *Especificación y Validación*, son similares a las ya descritas en el punto 2. Respecto a la fase de *Análisis*, en esta fase se representa la información colectada en un lenguaje ad-hoc y acorde al dominio del problema, a objeto de detectar y reducir las posibles ambigüedades, contradicciones, omisiones, o inconsistencias en los requisitos elicitados.

5. Utilización de técnicas de IR en el modelo propuesto

Se presenta a continuación una recomendación sobre qué técnica utilizar, en cada una de las fases del modelo propuesto, para la educción de requisitos en proyectos de Data Mining.

- *Comprensión del Negocio.* Ésta es la fase inicial del proceso, y tiene como tareas o actividades centrales, la interiorización en el dominio del problema, la creación de lazos de confianza, y la identificación de los problemas de negocio susceptibles de resolverse mediante técnicas de Data Mining. Estas actividades preferentemente deben realizarse, mediante la comunicación directa con clientes, usuarios y todos los interesados en el proyecto, por lo tanto, en esta fase se recomienda utilizar las siguientes técnicas:

  Entrevistas y cuestionarios. Pues, es la forma más simple de comunicación y permite la obtención de gran cantidad de información.

  Técnicas JAD y lluvia de ideas. Estas técnicas, son similares en sus características y procedimientos, mediante la comunicación entre todas las partes involucradas en el proyecto, permite la interiorización en el dominio del problema.

  Lectura de Documentación de Sistemas existentes. Si existen sistemas ya desarrollados, es posible que exista documentación sobre el dominio del problema.

- *Modelo de Negocio*. El modelo de negocios, deberá representar al negocio desde la perspectiva de la toma de decisiones. En su construcción, se podrán aplicar técnicas ad-hoc para este propósito,

tal como por ejemplo Laddering [15], [10], y una notación en lo posible estandarizada, como UML [3].

- *Modelo de Datos.* El modelo de Datos, es una abstracción que deberá contener información relativamente completa de la realidad objeto de estudio y en función de éste, corresponderá considerar su particular estructuración. Es posible construir un Data Mart ad-hoc al propósito del objeto de estudio, a partir de las bases de datos disponibles en la organización.
- *Obtención de Requisitos.* Esta fase tiene como objetivo central, obtener un conjunto de requisitos a partir de los cuales, se procederá al diseño de los modelos de Data Mining. Estos requisitos se pueden obtener directamente desde los stakeholders y también a partir del modelo de negocio. Las técnicas que se proponen son:

  Entrevistas y cuestionarios. Esta es una de las técnicas más ampliamente utilizadas para elicitar requisitos en todo tipo de proyectos. Una vez interiorizados sobre el dominio de negocio y creados los necesarios lazos de confianza, es posible establecer una fluida comunicación con los Stakeholders a fin de identificar los requisitos del proyecto.

  Casos de uso. Esta técnica por sus características, es adecuada para la identificación y elicitación de requisitos, pues la mayoría de los requisitos pueden ser expresados mediante casos de uso. Esto es además doblemente útil, pues una notación estándar también puede ser utilizada para representar el Modelo de Negocio.

  Técnicas JAD y lluvia de ideas. Estas técnicas, mediante la actividad de planteamiento de posibles soluciones, en forma implícita generan el conjunto de requisitos que se espera elicitar.
- *Análisis y negociación.* Estas actividades normalmente se llevan a cabo mediante la discusión entre las partes involucradas en el proyecto. Dadas las características de técnicas como lluvia de ideas y técnicas JAD, cuyo común denominador es la comunicación fluida que se logra entre las partes, su simplicidad y el hecho de facilitar el consenso, son las recomendadas para aplicarse en esta fase.
- *Especificación de requisitos.* La fase de especificación de requisitos, corresponde al proceso en el cual, se genera un acuerdo documentado entre las partes involucradas en el proyecto, respecto del problema a ser resuelto. Para la especificación de requisitos, las técnicas más recomendables son, casos de uso, técnicas JAD y lluvia de ideas. Los casos de uso, facilitan la representación no ambigua de los requisitos, las técnicas lluvia de ideas y JAD en su fase final de documentación, tienen como meta, la redacción de un documento con los principales acuerdos logrados.
- *Validación*. Las técnicas que mejor se adaptan para esta tarea son las técnicas JAD, que en su etapa de producción del documento final considera reuniones de revisión y validación, la técnica de análisis jerárquico que permite identificar y resolver los posibles conflictos que se puedan originar entre diversos requisitos, también es deseable en esta fase, la creación de modelos de prueba con el propósito de mostrarle al usuario cuales son los resultados concretos que se obtienen de un proyecto de Data Mining.
- *Documento de Requisitos.* Finalmente el documento de requisitos, puede ser construido apoyado por estándares tales como el estándar IEEE-830, o la plantilla VOLERE.

6. Conclusiones

Afirmaciones muy conocidas como *"no se puede dar en el blanco, si no se sabe donde esta el objetivo"*, o *"los proyectos sin metas claras, claramente no alcanzaran sus metas"*, grafican contundentemente la necesidad de conocer anticipadamente los requisitos de un producto antes de iniciar su construcción. Un proyecto de Data Mining, no escapa a esta regla. Considerando entonces la trascendencia que reviste la obtención de requisitos completos, detallados, entendibles y no ambiguos para el logro de un proyecto exitoso, esta tarea se puede realizar de mejor manera, utilizando algún tipo de proceso ordenado. Para obtener este orden, se debe diseñar el proceso, apoyándose en algún modelo que sirva de guía a la hora de definir, diferenciar y secuenciar las actividades.

Un proyecto de Data Mining, se circunscribe esencialmente, a la comprensión y exploración de los datos y por lo tanto, los requisitos deben estar enfocados a determinar, de qué manera los datos influyen en la toma de decisiones o de qué forma impactan en las decisiones que se toman. Para el logro de estos objetivos, es preciso entonces no desvincular

las actividades típicas de un modelo de proceso de Ingeniería de Requisitos, con modelos que representen el negocio desde una particular visión y de los datos que son el núcleo del proceso.

Hechas las consideraciones anteriores, en el presente trabajo se ha propuesto un modelo de proceso orientado a la educción de requisitos en proyectos de Data Mining, este modelo debe orientar y permitir la obtención de los requisitos de una manera organizada y sistemática, permitiendo además, el validar los requisitos en función de los modelos de negocio y de datos, en forma previa a la redacción del documento de contrato final.

Normalmente, los procesos de toma de decisiones, no son procesos bien estructurados, presentando la dificultad inherente de modelarlos. Por tanto los mayores esfuerzos deben enfocarse en la comprensión del dominio del negocio y problema, para posteriormente diseñar el modelo de negocios y consecuentemente el modelo de datos que permitan sustentar el proyecto de Data Mining.

Finalmente en una segunda etapa del proyecto de investigación en curso, se aplicará el proceso de Ingeniería de Requisitos propuesto, en varios proyectos de Data Mining de diferente tamaño, a objeto de validar el modelo y efectuar los ajustes que sean pertinentes, definiendo además previamente, las métricas necesarias para dicha validación.

Referencias

[1] Bahamonde J.M., Rossel R. "Un Acercamiento a la Ingeniería de Requerimientos", Universidad Técnica Federico Santa María, 2003.

[2] Chapman P., (NCR), Clinton J., (SPSS) Kerber R., (NCR), Khabaza T. (SPSS), Reinartz T. (DaimlerChrysler), Shearer C. (SPSS), and Wirth R. (DaimlerChrysler). "CRISP-DM 1.0 step-by-step data mining guide". Thechnical report, 2000.

[3] Eriksson H. E., "Bussiness Modeling with UML, Bussines Patterns at Work", Wiley Computer Publishing, 2000.

[4] Dilauro, L. "What's nest in monitoring technology? Data Mining finds a calling in centers", mayo 2000.

[5] Gordijn Jaap "Value-based Requirements Engineering Exploring Innovative e-Commerce Ideas", VRIJE UNIVERSITEIT, 2003.

[6] Portal www.isixsigma.com, "consulta sobre metodología 6-Sigma" [en línea], disponible en: http://www.isixsigma.com/sixsigma/six_sigma.asp [Consulta: 23 de Junio de 2005].

[7] Kotonya G. and Sommerville I. "Requirements Engineering. Processes and techniques". USA. J. Wiley, 1998.

[8] Laudon K. C., "Sistemas de Información Gerencial, Organización y Tecnología de la Empresa conectada en Red", Ed. Prentice, 2002.

[9] Martínez de Pisón Ascacibar, F.J. "Optimización mediante técnicas de minería de datos del ciclo de recocido de una línea de galvanizado", Tesis Doctoral, Universidad de La Rioja, Servicio de Publicaciones, 2003.

[10] Milton, N., Portal http://www.epistemics.co.uk, "Repertory Grid Technique" [en línea], disponible en: http://www.epistemics.co.uk, 20 de noviembre de 2003, [Consulta: 18 de julio de 2005].

[11] Portal http://www2.noticiasdot.com, "consulta sobre noticias de actualidad" [en linea], disponible en http://www2.noticiasdot.com/publicaciones/2002/0702/0307/noticias0307/noticias0307-1.htm, [consulta: 29 de Abril de 2006].

[12] Richards D., "A Process Model for Requirements Elicitation", Department of Computing Division of Information and Communications Sciences Macquarie University, Sydney, Australia, 2000.

[13] Rilston, F., Paim, S., and Castro, J. "DWARF: An Approach for Requeriments Definition and Management of Data Warehouse Systems", 11th IEEE International Requirements Engineering Conference (RE'03), p-75, 2003.

[14] Robertson S. and Robertson J., "Mastering the Requirement Process", Ed. Addison –Wesley, 1999.

[15] Rugg, G., Homepage de Gordon Rugg, [en línea], disponible en: http://mcs.open.ac.uk/gr768/elicitation/methods/laddering/laddering.shtml, site designed by Ed. De Quincey, 2003, [Consulta: 18 de julio de 2005].

[16] Portal www.sas.com, "Descripción de la metodologia SEMMA", [en línea], disponible en: http://www.sas.com/technologies/analytics/datamining/miner/semma.html [Consulta: 19 de Abril de 2005].

[17] Ian Sommerville, "Ingeniería de Software", 6ta. Edición, Ed. Addison Wesley, 2002.

[18] Weber M. and Weisbrod J., "Requirements engineering in automotive development: Experiences and challenges". IEEE Software, pages 16 –24, Enero/Febrero 2003.

[19] META Group Research-Delta Summary, A. Zornes, "The Top 5 Global 3000 Data Mining Trends for 2003/04 Enterprise Analytics Strategies, Application Delivery Strategies", Delta, 2061, March 26, 2003.

Optimizing Lies in State Oriented Domains based on Genetic Algorithms

A. Zylberberg [1], E. Calot [1], J. Ierache [2,1], H. Merlino [1,3], P. Britos [3,1], R. García-Martínez [3,1]
1. Laboratorio de Sistemas Inteligentes. Facultad de Ingeniería. Universidad de Buenos Aires
2. Instituto de Sistemas Inteligentes y Enseñanza de la Robótica. Universidad de Morón
3. Centro de Ingeniería del Software e Ingeniería del Conocimiento. Escuela de Postgrado. Instituto Tecnológico de Buenos Aires
jierache@unimoron.edu.ar, {hmerlino, pbritos, rgm}@itba.edu.ar

Abstract

This paper explores the use of genetic algorithms in identifying effective lies for use in negotiations with incomplete information in State Oriented Domains (SOD's). The aim is to seek lies which are not just safe (i.e. they will not lead to a disgraceful outcome) but also yield the highest possible utility. We propose a representation of a goal, a mechanism to assess aptitude and suitable GA operators.

1. Introduction

In the simplest possible scenario for a non-trivial negotiation, agents will state their objectives and then work their way towards a satisfying solution [1]. Finding such a solution will probably include agreeing on a final situation that is acceptable to every agent, as well as dividing the work necessary to reach that situation. Thus different roles will be designed, and in general the roles will have different costs. Fair enough, the agent with the most onerous objective will play the most expensive role. Naturally, it is in an agent's best interest to achieve his goal at the lowest possible cost [2]. That is where an agent may be tempted into stating an objective that is different from his actual goal. Lies can be beneficial, inasmuch as pretending to have a cheaper goal might result in doing less work in the joint plan. Provided there exists a mechanism that will guarantee the simultaneity of the exchange of information on the goals, speculation with lies is not just greatly restricted but also dangerous, since the negotiation could result in a solution that will not satisfy the agent's true goal. Nevertheless, in a context where abiding by such a mechanism is not mandatory, there may come to occur that an agent will become acquainted with the other agent's goal before the latter learns his. Should this be the case, lying can be safe, for it is possible to ensure that the outcome will not be disgraceful. Moreover, the agent can focus on finding a lie which is not just safe but also yields the highest possible utility. Firstly we present the reader with a simplified version of state oriented domains (section 2) and a genetic representation of a goal (section 3). Then we discuss how to assess the satisfaction of a goal (section 4), generate lies (section 5), assess their aptitude (section 6) and improve them through sensible mechanisms of selection (section 7), crossover (section 8) and mutation (section 9). Finally, we debate improvements to the model proposed (section 10.1) as well as extrapolation of the concepts developed to contexts with less strict hypotheses (section 10.2).

2. A simplified version of State Oriented Domains

The State Oriented Domain model [3] is intended to address a very wide range of problems. However, for the sake of clarity, in this paper we will apply some restrictions, in order to be able to study how to use genetic algorithms to find effective lies without losing focus by having to analyze endless general cases. In section 8 we discuss the generalization of our concepts. Some of the restrictions that we apply to SOD's are:

- We will study negotiations involving only two agents. We will refer to the agents as “us” and “rival”, and use the subscripts U and E respectively.
- The rival agent will always declare his goal first, and will always be telling the truth. Then we will take into consideration all the aspects involved, such as current world state, the rival's goal and our goal, and produce a lie, which is what we will declare our goal to be.
- As regards plans, we will only study pure deals. More complex types of deals, such as mixed deals, semi-cooperative deals and multi-plan deals add little

to the general idea of finding effective lies, and thus will not be analyzed in depth.

2.1 State Oriented Domains

Accordingly, we define a State Oriented Domain (SOD) as a tuple < *S* , *P* , *c* > where:

S is the set of all possible world states.

P is the set of all possible plans. A plan $p \in P$ moves the world from one state in S to another. In each plan there are two roles, p_1 and p_2, each consisting of a series of actions. As a particular case, a role could be null.

c is a function $c: P \rightarrow (\mathbb{R}^+)^2$. For each plan *p* in *P*, c(*p*) is a pair of positive real numbers, the cost of each role in the plan. If a role is null, its cost is 0. The higher cost in the plan is noted c+, and the lower cost is called c-. As a particular case, both roles could have the same cost.

2.2 Goals, Plans, Deals and Utilities

Definition 2.2.1 A goal is a subset of *S*. For instance, an agent's goal is the set of all world states that satisfy him. Notation: G_U our goal, G_R our rival's goal, G'_U our lie. That is to say, what we declare our goal to be, G_J the joint goal. Namely, $G_J = G_U \cap G_R$

Definition 2.2.2 An encounter within an SOD < *S* , *P* , *c* > is a tuple < s_0 , G_U , G_R > such that $s_0 \in S$ is the initial state of the world, G_U is the set of all world states that satisfy us and G_E is the set of all world states that satisfy the rival. Note that G_U can actually be G'_U if we are lying. In fact, what we study in this paper is how to determine an effective G'_U to declare instead of G_U.

Definition 2.2.3 A stand-alone plan is a plan in which only one of the agents has an active role (i.e. does something). The set of all stand-alone plans is noted *A*. It follows that $A \subset P$.

Definition 2.2.4 A joint plan is a plan in which both agents have an active role. The set of all joint plans is noted *J*.

Definition 2.3.4 Given an initial world state s_0 and a goal *G*, the stand-alone cost is the cost of the stand-alone plan that moves s_0 to a state in *G* at the minimum cost. Namely: $l = \min_{a \in W} [c(a)]$. Where $W \subset A$ is the set of stand-alone plans which satisfy *G* (from the initial world state s_0). Notation: l_U our stand-alone cost, l_R our rival's stand-alone cost, l'_U our fake stand-alone cost.

Definition 2.2.5 The utility of an agent is the difference between the cost he would have to pay to reach his goal alone, and the cost of the role in the joint plan that the negotiation results in. If in a joint plan we are assigned the most expensive role, our cost is *c*+, and otherwise it is *c*-. Let then c_U be our cost, our utility is: $U_U = l_U - c_U$

Definition 2.2.6 Given an encounter < s_0 , G_U , G_R >, a deal is a joint plan $j \in J$ that moves the world from s_0 to a state $s_f \in G_J$.

Definition 2.2.7 A deal is individual rational if the utility of each agent it not negative.

Definition 2.2.8 A deal is pareto optimal if there does not exist another deal that is better for one of the agents and not worse for the other.

Definition 2.2.9 The negotiation set NS is the set of all deals that are both individual rational and pareto optimal.

3. Genetic representation

Finding an effective lie can be treated as an optimization problem, hence the use of genetic algorithms. In every genetic algorithms model [2], there is a population of individuals (namely, possible solutions to the problem) which get improved generation after generation.

3.1 What individuals are

Our problem is to find an effective lie, and each individual in our population must be a possible solution to the problem. Consequently, each individual will be a lie. A lie is a goal (it is not our *real* goal, but it is a goal), therefore it is a subset of *S*. Finding a convenient representation for an individual in our genetic algorithms model is then finding a convenient representation for subsets of S.

3.2 Representing subsets of S

If *G* is a subset of *S*, then each world state in *S* may or may not be in *G*. Therefore, if there are *n* elements in *S* (that is to say, the world has n possible states) then there are 2^n possible subsets of *S*. For instance, if S = {“the book is on the table”, “the dog is out”, “the window is open”} then there are 8 possible subsets of

S, that is to say, 8 possible goals or lies. A first approach could be enumerating the elements of *S* that are present in the subset. This can be achieved by finding a function that will map every element of *S* unequivocally to a natural number in [1;n]. Then a subset of *S* can be represented by a stream of *n* bits, where the i-th bit will be 1 if the i-th element of *S* is present in *G*, and 0 otherwise. However, this approach has two main difficulties. Firstly, *n* could be a very large number. The world does not even need to be too complex to have millions of possible states. For example, the permutation of only 10 elements consists of 10! = 3,628,800 possibilities, and that means individuals of 442 kilobytes. Secondly, finding an analytical function that maps every element of S unequivocally to a natural number in [1;n] might not be feasible. We could still make a list of all the possible states and assign a number to each item on the list, but the list can be extremely large and require too much memory. Another approach is to represent a subset of *S* as a condition. Then *G* will be the set of all the elements in *S* that satisfy the condition. This solves the problems mentioned for enumeration, at the cost of introducing additional complexity in the design of our algorithms.

3.3 Representing conditions

If *S* is the set of all the apartments that can be bought, a lie could then be: "*I want an apartment with 3 bedrooms"*. Then *G* would be the set of all the apartments in *S* that have 3 bedrooms. However, the condition could be more complex: "*I want an apartment with 3 bedrooms and 2 bathrooms and it must have a balcony, or at least a patio. Closets scare me, so I definitely don't want one"*. This last condition can be rewritten as: "*(3 bedrooms) and (2 bathrooms) and (a balcony or a patio) and (not a closet)"*. We then note that a complex condition can be expressed in terms of atomic conditions and logical operators. In defining the possible atomic conditions, caution must be exercised, since they must be sufficient to cover every subset of *S*.

3.4 Condition trees

We will represent a goal as a tree [3]. We will call such a tree a *condition tree*. A state s in S is also in G if and only if it satisfies the root of the tree (see section 4). Condition trees have 3 different types of nodes: cond, and, or. *Cond* nodes contain atomic conditions. A *cond* node is satisfied by a state *s* if *s* satisfies the atomic condition contained in the node. For example, if the atomic condition is "3 bedrooms" then the node will be satisfied only by every *s* which has 3 bedrooms. Every leaf in a condition tree is a c*ond* node, and every *cond* node in a condition tree is a leaf. Also, note that if a tree only has one node, then that node would be a *cond* node. *And* nodes and *or* nodes have children. An *and* node is satisfied if and only if all of its children are satisfied. An *or* node is satisfied if and only if at least one of its children is satisfied. *And* and *or* nodes should always have at least 2 children. In figure 1 we see an *and* node with 3 children. Note that every child of an *and* or *or* node can be any type of node, namely another *and* or *or* node (nested operators) or a *cond* node. It might seem not reasonable for a child of an *and* node to be another *and* node, since for example:

a and (b and c) = a and b and c

The same happens with *or* nodes:

a or (b or c) = a or b or c

Nevertheless, such relationships between nodes may be allowed for the sake of diversity. This is discussed in section 6. Finally, note that there is no need for *not* nodes, since "not" is a unary operator. Accordingly, every node will have a "not" flag, which if activated will invert the node's satisfaction value. If an *and* node has the "not" flag activated, it will be satisfied if and only if any of its children is not satisfied, since by De Morgan's law [4]:

not (a **and** b) = **not**(a) or **not**(b)

If an *or* node has the "not" flag activated, it will be satisfied if and only if none of its children are satisfied, since by De Morgan's law [4]:

not (a **or** b) = **not**(a) and **not**(b)

If a *cond* node has the "not" flag activated, it will be satisfied if and only if the state does not satisfy the atomic condition it contains. In figure 2 we depict the condition tree for the second example of section 3.3.

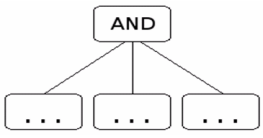


Figure 1. An *and* node with 3 children

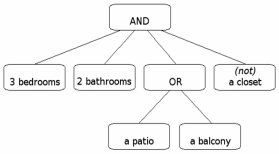


Figure 2. Condition tree for second example in 3.3.

3.5 Implementation

All nodes must have a "not" field, all nodes must have a "type" field; *and* and *or* nodes must have a list of pointers/references to nodes; *cond* nodes must have an atomic condition. An atomic condition is typically an instance of a class or struct having fields like: type

of condition (e.g. 1="has", 2="has a surface of", 3="has a price of"), numerical parameters (e.g. "number", "amount"), modifiers (e.g. "is exactly", "is less than", "is equal or greater than"), etc. (e.g. for the condition type 1 in this example, a field indicating 1="bedroom", 2="bathroom", 3="patio", etc.). As we said above the atomic conditions defined must be sufficient to make it possible to represent any goal an agent may have. For example, if there are two elements in *S* such that the only difference between them is the color a room is painted, then there must be a way to specify the color of rooms in an atomic condition.

4. Assessing the satisfaction of a goal

When we implement our genetic algorithm, we will need a function that determines whether a given state *s* in *S* satisfies a given goal *G*, that is to say, whether s is also in G. In section 3.4 we said that a state *s* in *S* is also in *G* if and only if it satisfies the root node of the condition tree for *G*. The function we are to implement will be recursive. Its parameters will be a state and a reference/pointer to a node. When it is necessary to determine whether *s* is in *G*, the function will be passed *s* and a reference/pointer to the root of the condition tree for *G*. The function will behave differently depending on the type of the node it receives:

- If it is a *cond* node, it will return true or false depending on whether the state satisfies or not the condition.
- If it is an *and* node, it will call itself recursively for every child of the node. Then if all the calls return true, it will return true. Else, it will return false. An optimization can be made here: if any of the calls returns false, then the function can return false without performing the calls for the remaining children.
- If it is an *or* node, it will call itself recursively for every child of the node. Then if all the calls return false, it will return false. Else, it will return true. An optimization can be made here: if any of the calls returns true, then the function can return true without performing the calls for the remaining children.

5. Generating the initial population

Unless for some reason we are able to determine a convenient set of data for the initial population, normally we will resort to simulation [2]. Simulation consists of creating random individuals whose parameters follow certain distributions [5]. A distribution is a set of pairs of possible values and associated probabilities, for example: A 30%, B 50%, C 20%. In section 5.1 we explain how to simulate a value, in section 5.2 we discuss which values should be simulated, and in section 5.3 we discuss the distributions and their effects.

5.1 Simulating a value

Typically, a random number function will be available. Without loss of generality, we will assume the function generates random numbers in the interval [0;1). That is to say, R is a random variable with a uniform(0;1) distribution. Let X be the value we want to simulate, $P_X(x)$ will relate the possible values to their probabilities. From that function we can obtain the cumulative distribution function:

$$F_X(x) = P(X \leq x) = \sum_{-\infty}^{x} P_X(x) \cdot$$

This function will start with value 0 at $-\infty$ and have at every possible value of X, where the height of the leap will be the probability of that value. The simulated values of X will be $F_X^{-1}(r)$, where r are values obtained from the random function. In figure 3 we show $F_X^{-1}(r)$ for A, 30% ; B, 50% ; C, 20%.

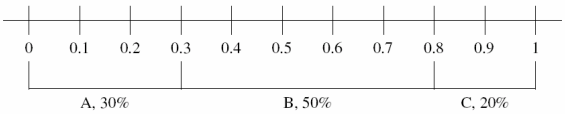


Figure 3. Values of $F_X^{-1}(r)$ for A, 30% ; B, 50% ; C, 20%.

In the figure 3 we can appreciate how the values of the random function are divided between A, B and C, according to their probabilities. If we get 0.62393 from the random function, the simulated value will be B. If we get 0.29218, the simulated value will be A. The pseudo-code would be:

```
generate random value;
if (random value < 0.3) return A;
else if (random value< 0.3+0.5) return B;
else return C;
```

5.2 What will we be simulating?

Before any analysis is carried out, we would like to advice against hard-coding the probabilities for the simulations inside the program's main code. Instead, they can be conveniently placed with constants or *#define*'s in a separate header or configuration file. We will now go through the process of simulating an individual. We will need a function called CreateNode. It will be a recursive function. This function will first

decide randomly which type of node it will generate, according to the probabilities defined. If it decides it will generate a *cond* node, then it must generate an atomic condition (see below) and return. If it decides to generate an *and* or an *or* node, then it must generate the number of children, and then call itself recursively to generate each child. Generating an atomic condition consists basically of applying the principles and method of simulation to each of the fields that describe a condition for the particular problem we are solving. For instance it will first decide randomly what type of condition it will be, and then simulate all the values for that type of condition.

5.3 Effects of the probability distributions

Keeping the actual probabilities used outside the program's code allows for easier experimentation. We recommend doing so. Experimenting with these parameters can give valuable information about the problem. Using a low probability for *or* nodes and a high probability for *and* nodes will make the initial lies more less specific and more vague, whereas using a high probability for *or* nodes and a low probability for *and* nodes will make the initial lies very restrictive. The latter might not be a good idea because in the beginning we want to broaden our options. Using a low probability for *cond* nodes will make the initial population complex, as the trees will go up several levels. If you intend to favor simple answers in the beginning of the search, you should use a high probability for *cond* nodes. Experimenting in various domains, we have found the following probabilities to yield, in our opinion, quite reasonable results: #define prob_and_node 0.1, #define prob_or_node 0.2, #define prob_cond_node 0.7. Be aware that any values you use are very strongly domain-dependent. Use the numbers given if you do not know where to start.

6. Aptitude assessment

The evolution mechanisms rely entirely on the aptitude of individuals, hence the cruciality of its computation. Note that it is extremely important to make sure that the lie we tell does not result in a negotiation that will lead to a disgraceful final world state, that is to say, a state that does not satisfy our *real* goal G_U. This is studied in section 6.4. However, for now we will assume that none or our lies can be disgraceful. If we say our goal is G'_U, and an agreement is reached, then c'_U will be our cost in the joint plan. We will define the aptitude of an individual G'_U to be $E[c'_U]$, that is, the expected value [5] of our cost in the joint plan provided we state our goal is G'_U. We are therefore going to minimize $E[c'_U]$. We could as well define the aptitude to be $E[c_U - c'_U]$ and maximize, but c_U is constant and also we would be getting negative values. Note that working only with positive values does not keep us from checking our results for individual rationality. Before we continue, we would like to point out that we are computing the expected value of the costs instead of the costs themselves because there may be the case where the agents have to toss a coin to decide who will play each role. In that event, the cost actually becomes a random variable [5]. The problem is then computing $E[c'_U]$ for a given lie G'_U. It follows that we will need to determine (or guess, if the rules are not so clear) how the negotiation would go were the rival to state his true goal G_R and us our fake goal G'_U. The negotiation mechanism will determine a pair of roles which are pareto optimal, and then assign each of them to an agent, based on the standalone costs and individual rationality.

6.1 Computing $E[c'_U]$

If the standalone cost of any of the agents is less than c, then the negotiation will not be individual rational. In that case, our apparent cost will equal our standalone cost, for we will have to do the work on our own (unless we are forced to cope with the existence of the other agent, this of course depending on the negotiation rules). But our real final cost will be l_U, because if we have to do the work alone we will probably pursue our real goal. If the standalone costs of both agents is no less than c-, then both agents are interested. Note that if the standalone cost of an agent is equal to c-, then he will agree to negotiate just to help the other agent, therefore achieving pareto optimality. If the standalone costs are the same, then a fair coin will be tossed to determine which role will be played by each agent. Our expected cost is $E[c'_U] = ½ . c- + ½ . c+ = (c- + c+) / 2$. If the costs of the roles are equal, then there is no need to toss the coin, and $c'_U = c- = c+$. In fact if the costs of the roles are equal, it never matters which role is played by each agent. If the standalone costs are different, and the costs of the roles are different, then the analysis becomes more complex. The utility available to be shared between the agents is the difference between the sum of the standalone costs and the sum of the costs of the roles. If the difference between the standalone costs is greater than the utility available to share, then the agent with the higher standalone cost must get the most expensive role (i.e. c+). Otherwise, a coin will be tossed, with a

probability such that the utility available is shared equally between the agents. Our expected cost is then:

$E[c'_U] = p \cdot c- + (1-p) \cdot c+$ *[6.1.a]*

Let *u* be the utility available to be shared. Let p be the probability that the coin favors us (i.e. we get c-), it is computed as follows. Since the utility must be shared equally between the agents, our expected utility $E[U'_U]$ must equal half that utility. Therefore we get:

$p \cdot (l'_u - c-) + (1-p) \cdot (l'_u - c+) = u / 2$ *[6.1.b]*

and we solve for p:

$$p = \frac{2c+ - 2l_U' + u}{2(c+ - c-)} \quad [6.1.c]$$

If then we plug *[6.1.c]* in *[6.1.a]* we get:

$E[c'_U] = l'_U - u / 2$ *[6.1.d]*

Now, the pseudo code for the above mechanism:

```
if (c'U < c- OR cR < c-) then return lU

if (l'U = lR) then return (c- + c+) / 2
if (c- = c+) then return c-
u = l'U + lR – c- – c+
if abs(l'U – lR) > u then:
   if (l'U > lR) return c+ else return c-
return l'U - u / 2
```

We have studied how to compute our expected cost. As we have showed, we will need to compute beforehand the standalone costs and the costs of the best joint plan. We will now focus on that.

6.2 Computing a standalone cost

For this task we use a recursive function. Its basic parameters are a world state, a goal and a counter to keep track of depth.

It is first called with the initial state of the world s_0. For each operation the agent can perform given that world state, it calls itself with the hypothetical world state that would result from carrying out that operation, and increasing the counter parameter. When the state satisfies the goal (see section 4), the function simply returns the counter's value. Every instance of the function that does not satisfy the goal returns the best value from the calls it made. This guarantees that when the first instance returns, the minimum standalone cost will be returned. In order to prevent infinite recursion, a maximum depth must be established. If the function is called with a counter value that exceeds the maximum, it returns “infinity”. Both the maximum and the “infinity” values should be defined outside the program's body using constants. Also note that you may need to include additional parameters to this function, to store information not reflected in the state of the world. For instance, if your definition of a world state includes the positions of blocks on a table but it does not take into consideration the blocks an agent might be carrying, then after a “pick up block” operation, the next recursive call should include the information that the agent is carrying a block. However, we advice you do include all the relevant information in your definition of world state. A performance optimization that should be considered for recursive functions like the one suggested here is the use of dynamic programming. It basically consists of adding memory to the recursive function, so that every instance will, prior to performing its normal computation, look up the received state on a list to check if it has been computed before. This is a trade between memory usage and speed.

6.3 Determining the costs of the roles of the best joint plan

This task is very similar to the previous one. The relevant differences are:

- The goal it receives is $G_J' = G_U' \cap G_R$. This can be implemented by creating an ***and*** node with 2 children, G'_U and G_R, and passing this new node to the function.
- Now it will try all the possible operations for role A and role B.
- Consequently, it will not have one but two counters: one for the number of operations in role A and another for the number of operations in the role B.
- Thus, it will not return a number but a pair of numbers.

6.4 Making sure the lie is not disgraceful

A lie is said to be disgraceful if it results in a negotiation that will lead to a final world state which does not satisfy our real goal G_U. Provided the negotiation mechanism is clear and unequivocal, then it is always possible to know whether a lie is disgraceful or not. However, that is hardly ever the case. For instance, say G_J' can be achieved by a joint plan with 2 roles of cost 2 each, leading to a satisfactory (i.e. not disgraceful) result. If the negotiation mechanism does not even guarantee that a joint plan with 2 roles of cost 2 each will be found, then definitely any lie could turn out to be disgraceful, since *anything* can happen. But note that even if the mechanism guarantees that a joint plan with 2 roles of cost 2 each is found, a lie could still be disgraceful, because the negotiation would simply lead to *any* final state that can be reached by some joint plan having 2 roles of cost 2 each. That is to say, not necessarily to

the final state we wanted, but actually to any final state that can be reached at the same cost. Therefore, for the negotiation mechanism to be safe, it needs to be clear and unequivocal, namely given an initial state and a joint goal it must always lead to the same final state. And like we said before, that could not be the case. So how do we make sure we never tell a lie that could turn out to be disgraceful?.We can start by pointing out that we will never accept a final state which is not in G_U', and the rival will never accept a final state which is not in G_R. Consequently, the final state is necessarily in $G_J' = G_U' \cap G_R$. In figure 4 we can see how easily a lie can be disgraceful, if it is in G_J' but not in G_U. However, we can choose to only tell lies such that no world states which satisfy the fake joint goal but not our real goal exist. In figure 5 we illustrate that since the final state will necessarily be in $G_J' = G_U' \cap G_R$, if we only tell lies such that $\overline{G_U} \cap G_R \cap G_U' = \varnothing$ then we could never end up in a disgraceful world state.

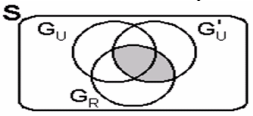


Figure 4. A disgraceful lie.

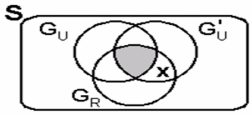


Figure 5. A graceful lie.

This could seem excessively restrictive, inasmuch as it might make many interesting results impossible. But in countless situations it is the only possible or reasonably simple way to guarantee that an acceptable final state will be reached.

7. Selection

There is nothing in particular to say about selection methods when it comes to this application. Any means of selection could be appropriate. Basically, we advice to experiment and determine the best method for the domain being used.

8. Crossover

8.1. Considerations

In general, a good crossover method should produce children that are similar to their parents. Otherwise, it becomes similar to random generation. Hence the need for a means to measure the similarity of any two goals. Therefore, it is necessary to define a metric so that distance between goals can be assessed. Any goal can be represented as a boolean function of the conditions its tree has. And boolean functions can be expressed with minterms and maxterms. We will define the distance between two goals as the number of minterms they differ in.

8.2. Definitions

Definition 8.2.1 The *distance* between two goals G_0 and G_1 is:

$$d(G_0, G_1) = \#((G_0 \cup G_1) - (G_0 \cap G_1))$$

where # represents the number of minterms in a boolean function having as variables the conditions in G_0 or G_1 or both. Note that this definition is also equivalent to:

$$d(G_0, G_1) = \#(G_0 \cup G_1) - \#(G_0 \cap G_1)$$

Definition 8.2.2 A *valid cross* between two goals is a linear combination of those two goals. This means that the minterms that are present in both parents must be present in the child, and that the minterms that are not present in either parent must not be present in the child. Children may or may not have the minterms that are present in only one of the parents.

Definition 8.2.3 The set of all *valid children* of the goals G_0 and G_1 is C_{G_0,G_1} We can imagine a linear function C = mX + b, there the slope is $m = G_0 \cup G_1$ and the C-intersect is $b = G_0 \cap G_1$. Then by using different values of X we get the set of valid crosses. This form allows for easy comprehension that:

$$C_{G_0,G_1} = \{(X \cap (G_0 \cup G_1)) \cup (G_0 \cap G_1) \forall X \subseteq S\}$$

which can be simplified to:

$$C_{G_0,G_1} = \{(X \cap G_0) \cup (X \cap G_1) \cup (G_0 \cap G_1) \forall X \subseteq S\}.$$

We will now show that $m = G_0 \cup G_1$ and $b = G_0 \cap G_1$ are in fact the slope and the C-intersect. Figure 6 illustrates definition 8.2.3. Definition 8.2.2 forces every child to have the minterms present in both parents. These minterms are the minterms present in the intersection of the goals. Therefore, by using $b = G_0 \cap G_1$ we guarantee that the common minterms will be added to every valid child. Definition 8.2.2 also implies that children may have any minterm as long as it is present in one of the parents. Nevertheless, note that since the minterms present in both parents are always present in the children, then this condition can be simply expressed as: children may have any minterm as long as it is present in at least one of the parents, namely $G_0 \cup G_1$. Thus, being the slope *m* that union, *X* will make some of the minterms present in at least one of the parents present in the child.

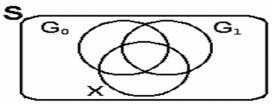


Figure 6. Ilustration of definition 8.2.3

Finally, note that since the minterms that are not present in either parent are not present in *m* or *b*, they are never present in the children. It is easy to see that:

1.- $X \equiv \varnothing \Rightarrow c \in C_{G_0,G_1} = G_0 \cap G_1$

2.- $X \equiv S \Rightarrow c \in C_{G_0,G_1} = G_0 \cup G_1$

3.- $X \equiv G_0 \Rightarrow c \in C_{G_0,G_1} = G_0$

4.- $X \equiv G_1 \Rightarrow c \in C_{G_0,G_1} = G_1$

In spite of the fact that 3 y 4 are valid crosses, they are not effective because the child is identical to one of the parents. That would be selection rather than crossover. As regards 1 and 2, they are valid, effective and easy to implement crosses. Performing intersections and unions alternatively is the simplest way to have a powerful crossover mechanism. However, one may argue that using only intersections and unions might hinder diversity. Consequently, we will now study other options for *X*.

Let $K_{G_0,G_1} \subseteq C_{G_0,G_1}$ be the subset of all computationally feasible crosses. We are only interested in crosses that are in K. Let G_0 and G_1 be two trees with *or* and *and* root nodes respectively. They can be expressed as $G_0 = \sum_{\forall n} S_n$ and $G_1 = \prod_{\forall n} R_n$. If we then partition the S conditions in two sets A and B, and the R conditions in two sets C and D, we get: $G_0 = A + B$ and $G_1 = CD$. The problem is them reduced to crossing A+B and CD. We will now use the linear function and a seed *X* to obtain valid crosses. Plugging G_0 and G_1 in definition 8.c we get:

$$C_{A+B,CD} = \{(X \cap (A+B)) \cup (X \cap CD) \cup ((A+B) \cap CD) \forall X \subseteq \zeta\}$$

$$C_{A+B,CD} = \{(X(A+B)) + (XCD) + ((A+B)CD) \forall X \subseteq \zeta\}$$

Simplifying:

$$C_{A+B,CD} = \{X(A+B+CD) + (A+B)CD \forall X \subseteq \zeta\}$$

Finally, it suffices to choose X conveniently to get elements in K.

$$X = A \Rightarrow c \in K_{A+B,CD} = A(A+B+CD) + (A+B)CD = A + BCD$$

$$X = B \Rightarrow c \in K_{A+B,CD} = B(A+B+CD) + (A+B)CD = B + ACD$$

$$X = C \Rightarrow c \in K_{A+B,CD} = C(A+B+CD) + (A+B)CD = C(D+A+B)$$

$$X = D \Rightarrow c \in K_{A+B,CD} = D(A+B+CD) + (A+B)CD = D(C+A+B)$$

These crosses are not just computationally feasible but also effective because they involve all four A, B, C, D. Using the same procedure it is possible to obtain:

- Children which are closer to G_1 because they do not depend on either A or B:
  {C(D+B);C(D+A);D(C+A);D(C+B)}
- Children which are closer to G_0 because they do not depend on either C or D:
  {A+BC;A+BD;B+AC;B+AD}

9. Mutation

When we want a mutated individual *G'* to be similar to the original individual G, the simplest way is to perform $G' = G \cup W$, where W is a minterm based on the conditions present in G. For a greater variation, we can use as W a portion of a minterm, that is to say, the intersection of some of the conditions present in G, some of which will have a not operator applied. For a complete variation, we can always resort to generation, as described in section 5.

10. Conclusions

We have explored in this paper how genetic algorithms can be used to identify effective lies for use in negotiations with incomplete information in State Oriented Domains (SOD's) seeking lies which are not just safe but also yield the highest possible utility. The proposed model has been implemented and testing is being carried out. Research in lie optimization is a novel area so there are not many results for comparison. Ethical considerations have been put a side to show a naïve way in which autonomous intelligent systems may lie during an automated negotiation process.

11. References

[1] García Martínez, R. y Borrajo, D.2000. *An Integrated Approach of Learning, Planning and Executing*. Journal of Intelligent and Robotic Systems 29(1):47-78.

[2] García-Martínez, R., Borrajo, D., Britos, P. y Maceri, P. 2006. *Learning by Knowledge Sharing in Autonomous Intelligent Systems.* Lecture Notes in Artificial Intelligence, 4140: 128-137.

[3] Zlotkin, G. and Rosenschein, J. 1996. *Mechanisms for Automated Negotiation in State Oriented Domains*. Artificial Intelligence, 86(2): 195-244.

[4] Grimaldi, R. 1989. *Discrete and Combinatorial Mathematics*, 2nd ed., Addison-Wesley.

[5] Zylberberg, A. 2005. *Probabilidad y Estadística*. Editorial Nueva Librería.

Extensión del Lenguaje SQL con Nuevas Primitivas SQL para el Descubrimiento de Reglas de Clasificación

Ricardo Timarán Pereira
Universidad de Nariño, Departamento de Ingeniería de Sistemas
San Juan de Pasto, Colombia
ritimar@udenar.edu.co

Resumen

La gran mayoría de sistemas de DCBD son débilmente acoplados con un SGBD. Sus principales desventajas son la escalabilidad y el rendimiento. Un Sistema DCBD fuertemente acoplado con un SGBD resuelve estos problemas. En este artículo, se muestran los primeros resultados para la tarea de clasificación del proyecto PostgresKDD, cuyo objetivo es dotar al SGBD PostgreSQL de la capacidad de descubrir conocimiento. En él se aplica el método tres-pasos para integrar la tarea de clasificación al interior de un SGBD. Se definen nuevos operadores algebraicos relacionales que faciliten la tarea de clasificación, se extiende el lenguaje SQL con nuevas primitivas para soportar esta tarea y finalmente se implementan en PostgreSQL El resultado es un sistema DCBD fuertemente acoplado con un SGBD.

Keywords: *Knowledge Discovery in Databases, New SQL Primitives for Data Mining, New Relational Algebraic Operators, Classification Rules, PostgresKDD*

1. Introducción

Muchos investigadores [27] [3] [1] [8] [10] han reconocido la necesidad de integrar los sistemas de descubrimiento de conocimiento y bases de datos, haciendo de esta una área activa de investigación. Los enfoques de integración de sistemas de Descubrimiento de Conocimiento en Bases de Datos (DCBD) y Sistemas Gestores de Bases de Datos (SGBD), reportados en la literatura, se pueden ubicar en uno de tres tipos de arquitectura: sistemas débilmente acoplados, medianamente acoplados y sistemas fuertemente acoplados [28].

La gran mayoría de Sistemas de DCBD son débilmente acoplados con un SGBD. La ventaja de esta arquitectura es su portabilidad. Sus principales desventajas son la escalabilidad y el rendimiento. El problema de escalabilidad consiste en que las herramientas y aplicaciones de este tipo de arquitectura, cargan todo el conjunto de datos en memoria, lo que las limita para el manejo de grandes cantidades de datos. El bajo rendimiento se debe a la copia de registros de la base de datos a la aplicación y al alto costo de las operaciones de entrada/salida cuando se manejan grandes volúmenes de datos. Un Sistema DCBD fuertemente acoplado con un SGBD resuelve los problemas de escalabilidad y rendimiento de las otras arquitecturas, debido a que todos los algoritmos son ejecutados conjuntamente con los datos en el SGBD.

Hay propuestas de investigación que discuten la manera como tales sistemas pueden ser implementados: expresando ciertas operaciones de minería de datos como una serie de consultas SQL [33] [35] [34], diseñando e implementando nuevos lenguajes de consulta como extensiones del lenguaje SQL con nuevos operadores unificados, los cuales soportan ciertas tareas de minería de datos: *DMQL* [9], *MSQL* [11], *Mine Rule* [16], *OLE DB for Data Mining* [17] [19][20], *SQL/MM* [23] [15], y definiendo nuevas primitivas SQL genéricas que facilitan el proceso de DCBD sin soportar una tarea en particular: *NonStop SQL/Mx* [4] [5], *Count by Group*, *Count by Order Group* [6] [7], *FilterPartition*, *ComputeNodeStatics*, *PredictionJoin* [25].

En este artículo, se muestran los primeros resultados para la tarea de clasificación del proyecto PostgresKDD, aprobado por el Sistema de Investigaciones de la Universidad de Nariño (Colombia), cuyo objetivo es dotar al SGBD PostgreSQL de la capacidad de descubrir

conocimiento. Igual que con la tarea de asociación, se utiliza el *método tres-pasos* propuesto en [29] [30] [31] para integrar la tarea de minería de datos clasificación al interior de un Sistema Gestor de Base de Datos Relacional (SGBDR). Este método consiste en que primero se extiende el álgebra relacional con nuevos operadores algebraicos que faciliten los procesos computacionalmente más costosos de la tarea de minería de datos. En el segundo paso, se extiende el lenguaje SQL con nuevas primitivas que se expresan en la cláusula SQL SELECT y que implementan los nuevos operadores algebraicos y en el tercero, se unifican estas primitivas en un nuevo operador SQL, en una nueva cláusula, que extrae el conocimiento deseado por la tarea de minería de datos. El resultado es un sistema DCBD fuertemente acoplado con un SGBDR con la capacidad de descubrir reglas de Clasificación.

El resto del artículo se organiza de la siguiente manera. En la sección 2, se describen los trabajos relacionados con las nuevas primitivas propuestas. En la sección 3, se presenta una breve introducción sobre la tarea de clasificación En la sección 4, se definen los nuevos operadores con los cuales se extiende el álgebra relacional para soportar la tarea de clasificación. En la sección 5 se describen las nuevas primitivas con las que se extiende el lenguaje SQL para la tarea de clasificación. En la sección 6, se presenta un nuevo operador unificado SQL para la extracción de reglas de clasificación. En la sección 7 se presentan los resultados de la implementación y pruebas de las primitivas en el SGBD postgreSQL y finalmente, en la última sección se presenta las conclusiones.

2. Trabajos relacionados

A pesar del acelerado avance y del incremento de la investigación en el área de Descubrimiento de Conocimiento en Bases de Datos, relativamente son pocas las propuestas de integrar, al lenguaje de consultas SQL, nuevas primitivas que permitan descubrir conocimiento eficientemente en grandes volúmenes de datos. Específicamente para la tarea de clasificación se han hecho las siguientes propuestas:

Han et al [9], proponen DMQL (Data Mining Query Language), un lenguaje de consulta para minería de datos en bases de datos relacionales. Su ventaja es su poder para expresar diferentes operaciones de minería de datos , entre ellas clasificación, con una sintaxis *SQL-like* (azúcar sintáctico) [2], pero carecen de una semántica formal en el álgebra relacional que soporte estas operaciones y por consiguiente, no se pueden aplicar técnicas de optimización para consultas de minería de datos. Por otra parte, este lenguaje se ha implementado en una arquitectura que no es fuertemente acoplada. D*MQL* está implementado en el sistema *DBMiner* [8] [9], un sistema para minería de datos débilmente acoplado con SGBD relacional.

Microsoft Corporation propone un lenguaje de minería de datos denominado *OLE DB for Data Mining* [17] [19][20] como un esfuerzo hacia la estandarización de primitivas de los lenguajes de minería de datos. Las especificaciones de *OLE DB for DM* cubren las primitivas para la creación y uso de varios métodos de minería de datos, incluyendo asociación, clasificación, predicción y *clustering*. La desventaja es que en este lenguaje no se cuenta con un sistema fuertemente acoplado y por tanto, las herramientas de minería de datos están por fuera de los SGBD y se acoplan a ellos por medio de una API (Application Programming Interface) [2]. *OLE DB for DM* es un API diseñada, principalmente, para trabajar con el SGBD *SQL Server* al cual se pueden conectar las herramientas de minería de datos, de diferentes proveedores, que cumplan con las especificaciones del API *OLE DB for DM* [19][20].

La Organización Internacional de Estandares *ISO* (*International Organization Standardization*) basada en el SQL 3, desarrolló un nuevo estandar de SQL conocido como SQL/MM (SQL Multimedia) [15] [12] [13]. La parte 6 [13], [23],hace referencia a la minería de datos (SQL/MM *Data Mining*) en la cual se definen tipos estructurados de SQL definidos por el usuario , incluyendo métodos sobre los tipos, para el descubrimiento de información "escondida" en grandes cantidades de datos. SQL/MM DM soporta cuatro diferentes técnicas de minería de datos: *rule model, clustering model, regression model* y *classification model*. Para cada modelo, hay un tipo definido por el usuario, conocido como *DM_*Model* (donde * es reemplazado por *Clas* para el modelo de clasificación) [15]. Estos tipos se utilizan para definir el modelo que se quiere aplicar en la minería de datos.

SQL/MM DM provee una interfaz estandar para los algoritmos de minería de datos, la cual puede ser usada como una capa por encima de cualquier sistema de bases de datos objeto-relacional, y más aún, ser usada como un *middleware* si fuese necesario. Estas características hacen que un SGBDR no soporte a SQL/MM DM.

Freitas [6] [7] propone las primitivas *Count by Group* y *Count By Ordered Group*, como estructuras genéricas para evaluar reglas candidatas en algoritmos

de reglas de inducción (clasificación) y particularmente, en árboles de decisión que se ejecutan obligatoriamente en SGBD paralelos. *Count by Group* y *Count by Ordered Group* no se han implementado como nuevas primitivas SQL, éstas se expresan mediante cláusulas SQL GROUP BY, que se pueden, simultáneamente, ejecutar en SGBD paralelos [7]. Esta solución no sería eficiente si se implementa en un SGBD no paralelo.

Sattler y Dunemmann [25] proponen las primitivas *FilterPartition*, *ComputeNodeStatics*, *PredictionJoin*, como estructuras genéricas para soportar la construcción y aplicación de árboles de decisión para clasificación, basados en la extensión de las características actuales de los SGBD objeto-relacional (i.e. funciones de tabla definidas por el usuario, funciones agregadas definidas por el usuario, índices extendidos) y en el estándar SQL-99. Estas primitivas se han implementado en C, como funciones de tabla definidas por el usuario usando la interface de llamadas de *Oracle 8i* (OCI) [25], lo que la convierte en una arquitectura de acoplamiento mediano.

3. Reglas de Clasificación

La clasificación de datos es el proceso por medio del cual se encuentra propiedades comunes entre un conjunto de objetos de una base de datos y se los clasifica en diferentes clases, de acuerdo al modelo de clasificación. Este proceso se realiza en dos pasos: en el primer paso se construye un modelo en el cual, cada tupla, de un conjunto de tuplas de la base de datos, tiene una clase conocida (etiqueta), determinada por uno de los atributos de la base de datos, llamado *atributo clase*. El conjunto de tuplas que sirve para construir el modelo se denomina *conjunto de entrenamiento*. Cada tupla de este conjunto es un ejemplo de entrenamiento. En el segundo paso, se usa el modelo para clasificar. Inicialmente, se estima la exactitud del modelo utilizando un conjunto de tuplas de la base de datos, generalmente diferente al de entrenamiento, cuya clase es conocida, denominado *conjunto de prueba*. A cada tupla de este conjunto se denomina ejemplo de prueba.

La exactitud del modelo, sobre el conjunto de prueba, es el porcentaje de ejemplos de prueba que son correctamente clasificadas por el modelo. Si la exactitud del modelo se considera aceptable, se puede usar para clasificar futuros datos o tuplas para los cuales no se conoce la clase a la cual pertenecen.

4. Nuevos Operadores del Álgebra Relacional para Clasificación

El modelo de clasificación basado en árboles de decisión, es, probablemente, uno de los más utilizados por su simplicidad y facilidad para entenderlo [25]. Entre los algoritmos de clasificación para árboles de decisión se cuentan ID-3 [21], C4.5 [22], SPRINT [26] y SLIQ [14]

El cálculo del valor de la métrica que permite seleccionar, en cada nodo, el atributo que tenga una mayor potencia para clasificar sobre el conjunto de valores del atributo clase, es la parte más costosa del algoritmo utilizado [35]. Un nuevo operador algebraico para clasificación basado en árboles de decisión debe facilitar el cálculo de las diferentes combinaciones de los atributos condición con el atributo clase y ofrecer funciones agregadas específicas, que permita el cálculo de estas métricas.

4.1 Operador Mate (μ)

El operador *Mate* genera, por cada una de las tuplas de una relación, todas los posibles combinaciones de los valores no nulos de los atributos de una lista de atributos denominados *Atributos Condición*, con el valor no nulo del *Atributo Clase*.

Formalmente, sea $A = \{A_1, \ldots, A_n\}$ el conjunto de atributos de la relación R de grado n y cardinalidad m, $LC \subset A$, $LC \neq \phi$ la lista de atributos condición a emparejar y n' el número de atributos de LC, $|LC| = n'$, $n' < n$, $Ac \in A$, $Ac \cap L = \phi$ el atributo clase con el que se emparejarán los atributos de LC. El operador μ aplicado a la lista de atributos LC, al atributo clase Ac de la relación R se define:

$$\mu_{LC;\,Ac}(R) = \{\, ti(M) \mid M = LC \cup Ac,\ LC \subset A,\ |LC| = n',\ n' < n,\ Ac \in A,\ Ac \cap LC = \phi,\ ti = Xi,\ 1 \le i \le m',\ m' = (2^{n'}-1)*m,\ \forall_i \forall_k (\ X_i = <null, \ldots, v_i(A_k) \ldots, null, \ldots, v_i(Ac)>,\ v_i(A_k)\ v_i(Ac) \neq null),\ 1 \le k \le n' \,\}$$

Ejemplo 1. Sea la relación $R(A,B,C,D)$ de la figura 1. Obtener las diferentes combinaciones de los atributos A,B con el atributo D, es decir, $R1 = \mu_{A,B;D}(R)$.

4.2 Operador agregado *Entro*

El operador agregado *Entro* permite calcular la entropía de una relación R con respecto a un atributo denominado atributo condición y un atributo clase.

Formalmente, sea $A=\{A_1, ..., A_n, Ac\}$ el conjunto de atributos de la relación *R* con esquema *R(A)*, extensión *r(A)*, grado *n* y cardinalidad *m*. Sea *t* el número de

A	B	C	D
a1	b1	c1	d1
a1	b2	c1	d2

A	B	D
a1	null	d1
null	b1	d1
a1	b1	d1
a1	null	d2
null	b2	d2
a1	b2	d2

Figura 1. Resultado operación $R1=\mu_{A,B;D}(R)$

distintos valores del *atributo_clase Ac*, $Ac \in R(A)$ que divide a *r(A)* en *t* diferentes clases, C_i $(1 \leq i \leq t)$. Sea r_i el número de tuplas de *r(A)* que pertenecen a la clase C_i. Sea *q* el número de destintos valores $\{v_1(A_k), v_2(A_k), .., v_q(A_k)\}$ del *atributo* condición A_k , $A_k \in R(A)$, el cual particiona a *r(A)* en *q* subconjuntos $\{S_1, S_2, \ldots S_q\}$, donde S_j contiene todos las tuplas de *r(A)* que tienen el valor $v_j(A_k)$ del atributo A_k. Sea s_{ij} el número de tuplas de la clase C_i en el subconjunto S_j. *Entro*$(A_k; Ac; R)$, retorna la entropía de *R* con respecto al atributo A_k, que se obtiene de la siguiente manera :

$$Entro(A_k;Ac;R)=\{y \mid y = -\sum p_{ij} \log_2(p_{ij}),\ 1 \leq i \leq t,\ 1 \leq j \leq q,\ p_{ij} = s_{ij} / |S_j| \}$$

donde $p_{ij} = s_{ij} / |S_j|$ es la probabilidad que una tupla en S_j pertenezca a la clase C_i.

La entropía de *R* con respecto al atributo clase *Ac* es:

$$Entro(Ac;Ac;R)=\{y \mid y = -\sum p_i \log_2(p_i),\ 1 \leq i \leq t,\ p_i = r_i / m\}$$

donde p_i es la probabilidad que un tupla cualquier pertenezca a la clase C_i y r_i el número de tuplas de *r(A)* que pertenecen a la clase C_i.

4.3 Operador agregado *Gain*

El operador agregado *Gain* permite calcular la reducción de la entropía causada por el conocimiento del valor de un atributo de una relació y se define:

$$Gain(A_k;Ac;R)=\{y \mid y = Entro(Ac;Ac;R) - Entro(A_k;Ac;R)\}$$

donde *Entro* $(Ac; Ac; R)$ es la entropía de la relación *R* con respecto al atributo clase *Ac* y *Entro*$(A_k;Ac;R)$ es la entropía de la relación *R* con respecto al atributo A_k .

4.4 Operador *Describe Classifier* (βμ)

Describe Classifier (βμ) es un operador unario, que toma como entrada la relación resultante de los operadores *Mate*, *Entro* y *Gain* y produce una nueva relación con los valores de los atributos que formarán los diferentes nodos del árbol de decisión.

Formalmente, sea $A=\{A_1, .., A_n, E, G\}$ el conjunto de atributos de la relación *R* de grado *n+2* y cardinalidad *m*. El operador βμ se define:

$$\beta\mu\ (R) = \{ t_i(Y) \mid Y=\{N,P,A,V,C\} \}$$

donde,

t_i=<*val(N),null,val(A),null,null*> si t_i es nodo raiz,

t_i=<*val(N),val(P),val(A),val(V),val(C)*> si t_i es nodo hoja

t_i=<*val(N),val(P),val(A),val(V),null*> si t_i es nodo interno

El operador βμ aplicado a *R*, produce una nueva relación con esquema *R(Y), Y={N,P,A,V,C }* donde *N* es el número del nodo, *P* identifica al nodo padre, *A* identifica el nombre del atributo asociado a ese nodo, *V* es un valor para el atributo *A* y *C* es el atributo clase. Su extensión *r (Y)*, está formada por un conjunto de tuplas en las cuales si los valores de los atributos son: $N \neq$null, *P*=null, $A \neq$null, *V*=null y *C*=null corresponde a un nodo raiz; si $N \neq$null, $P \neq$null, $A \neq$null, $V \neq$null y $C \neq$null corresponde a una hoja o nodo terminal y si $N \neq$null, $P \neq$null, $A \neq$null, $V \neq$null y *C*=null corresponde a un nodo interno.

Describe Classifier facilita la construcción del árbol de decisión y por consiguiente la generación de reglas de clasificación.

5. Nuevas Primitivas SQL para Clasificación

El nuevo operador algebraico *Mate,* conjuntamente con los nuevos operadores agregados *Entro()* y *Gain(),* extienden el álgebra relacional para soportar la tarea de clasificación. Para expresar estos operadores en el lenguaje SQL, es necesario implementarlos como nuevas primitivas SQL. De esta forma, el lenguaje SQL será capaz de soportar eficientemente esta tarea.

5.1 Primitiva *Mate By*

Esta primitiva implementa el operador algebraico *Mate* en la cláusula SQL SELECT. *Mate By* toma los valores de los atributos de una tabla denominados *atributos condición* y por cada registro forma todas las posibles combinaciones de estos atributos con otro atributo de la misma tabla denominado *atributo clase*. Este proceso lo realiza en una sola pasada sobre la tabla.

Dentro de la cláusula SELECT, *Mate By* tiene la siguiente sintaxis:

SELECT <*ListaAtributosTablaDatos*>
[INTO <*NombreTablaMate*>]
FROM <*NombreTablaDatos*>
WHERE <*CláusulaWhere*>
MATE BY<*ListaAtributosCondicion*> WITH <*AtributoClase*>
GROUP BY <*ListaAtributosTablaDatos*>

<*ListaAtributosTablaDatos*>::=<*Atributo*>,
<*ListaAtributos*>
<*ListaAtributos*>: :=<*Atributo*>, <*ListaAtributos*>
<*ListaAtributos*>: :=<*Atributo*>
<*ListaAtributosCondicion*>::=<*Atributo*>,
<*ListaAtributos*>
<*AtributoClase*>::=<*Atributo*>

Ejemplo 2. Sea la tabla SINTOMAS (SID,D_MUSCULAR,TEMPERATURA, GRIPA) . Obtener las ocurrencias de las diferentes combinaciones de los *atributos d_muscular, temperatura* con el atributo *gripa* y almacenar el resultado en la tabla *clasesintomas*. La orden SQL que permite obtener esta consulta es la siguiente:

SELECT *d_muscular,tempeartura,gripa, count(*)*
INTO *clasesintomas*
FROM *sintomas*
MATE BY *d_muscular,tempertura* WITH *gripa*
GROUP BY *d_muscular,tempertura,gripa*

Una vez se seleccionan los atributos especificados en la cláusula SELECT, la primitiva MATE BY los combina con los atributos d*_muscular,tempertura* y *gripa* . El resultado se agrupa con la cláusula GROUP BY, se cuenta la frecuencia de las parejas y finalmente, se almacena en la tabla *clasesintomas*.

5.2 Función agregada SQL *Entro()*

Esta función agregada SQL implementa el operador algebraico agregado *Entro()* en la cláusula SQL SELECT. SQL *Entro()* permite calcular, conjuntamente con la primitiva *Mate By*, la entropía de una tabla con respecto a cada una de las combinaciones de los *atributos condición* con el *atributo clase*. SQL *Entro()* se debe ejecutar conjuntamente con la función agregada *Count()*.

La sintaxis de SQL *Entro()* en de la cláusula SELECT es la siguiente:

SELECT <*ListaAtributosTablaDatos*>, *Count(*), Entro(*)* [INTO <*NombreTablaMate*>]
FROM <*NombreTablaDatos*>
WHERE <*CláusulaWhere*>
MATE BY<*ListaAtributosCondicion*> WITH <*AtributoClase*>
GROUP BY <*ListaAtributosTablaDatos*>

5.3 Función agregada SQL *Gain()*

Esta función agregada SQL, implementa el operador algebraico agregado *Gain()* en la cláusula SQL SELECT. SQL *Gain()* permite calcular, conjuntamente con la primitiva *Mate by*, la ganancia de información de una tabla con respecto a cada una de las combinaciones de los *atributos condición* con el *atributo clase*. Internamente SQL *Gain()* calcula la entropía del *atributo clase*, por ello se debe ejecutar conjuntamente con las funciones agregadas *Count()* y *Entro()*.

La sintaxis de *Gain ()* en la cláusula SELECT es:

SELECT <*ListaAtributosTablaDatos*>, *Count (*), Entro (*), Gain (*)*
[INTO <*NombreTablaMate*>]
FROM <*NombreTablaDatos*>
WHERE <*CláusulaWhere*>
MATE BY<*ListaAtributosCondicion*> WITH <*AtributoClase*>
GROUP BY <*ListaAtributosTablaDatos*>

6. Nuevo operador SQL para Clasificación

El operador SQL *Describe Classification Rules* implementa el operador algebraico *Describe Classifier* en una nueva cláusula SQL. Este operador construye el árbol de decisión y genera las reglas de clasificación. El operador *Describe Classification Rules*, permite la construcción del árbol de decisión de manera unificada con el cálculo de la métrica de particionamiento con la primitiva *Mate By* y las funciones agregadas *Entro()* y *Gain()* en una sola

instrucción SQL o en forma separada con sentencias SQL independientes, para luego generar las reglas.

Describe Classification Rules tiene la siguiente sintaxis:

DESCRIBE CLASSIFICATION RULES
[INTO <*TablaReglasClasificación*>]
FROM <*NombreTablaArbol*>
USING <*NombreTablaMétrica*>
[DO <*SubconsultaCálculoMétrica*>]

<*SubconsultaCálculoMétrica*>::=<SFWMG>
<SFWMG> ::= <SELECT FROM WHERE MATE BY GROUP BY>

Ejemplo 3. Sea la tabla CLIENTES(EDAD, INGRESOS,ES_ESTUDIANTE,MANEJOCREDITO, COMPRAEQUIPO) cuyos valores han sido discretizados. Encontrar las reglas de clasificación y almacenarlas en la tabla ReglasClasificación.

La sentencia SQL unificada que genera las reglas de clasificación es:

DESCRIBE CLASSIFICATION RULES
INTO *reglasclasificación*
FROM *nodosarbol*
USING *métricaspartición*
DO SELECT *edad, ingresos, es_estudiante, manejocredito, compraequipo, count(*), Entro() AS Entropia, Gain() AS Ganancia*
INTO *métricaspartición*
FROM *clientes*
MATE BY *edad, ingresos, as_estudiante, manejocredito* WITH *compraequipo*
GROUP BY *edad, ingresos, es_estudiante, manejocredito, compraequipo*

7. Implementación, pruebas y evaluación

Para evaluar el rendimiento de las primitivas implementadas, se utilizó un equipo Pentium IV de 64 bits a 3.0 Ghz, con 2GB de RAM, disco duro serial ata de 160GB y bajo sistema operativo Fedora Core 5 y la versión de PostgreSQL 7.3.4.

7.1 Aspectos de implementación

Con el fin de acoplar fuertemente la primitiva SQL *Mate by* al interior del motor PostgreSQL, se modificaron las estructuras, funciones y nodos en la capa intermedia de la arquitectura de este SGBD. Se modificó el *parser* para que construya, transforme y añada, a las estructuras del compilador, una lista de *atributos condición* y el *atributo clase*. En el *Planner/Optimizer* se agregó un nuevo nodo al *Query tree* denominado *Mate*. Se modificó el *Executor* para que sea capaz de generar por cada una de las tuplas, todas los posibles combinaciones formadas por los valores no nulos de los atributos pertenecientes a la lista de atributos condición y el valor no nulo del atributo clase.

Las funciones agregadas *Entro()* y *Gain()* y el operador *Describe Classifier Rules* se implementaron como funciones definidas por el usuario utilizando el lenguaje *plpgsql*.

7.2 Análisis de Resultados

Para las pruebas con la primitiva de clasificación, se utilizaron repositorios especializados para este tipo de tarea, disponibles en *http://www.ics.uci.edu/~mlearn/databases/*. Se tomo, el conjunto de datos : Zoológico (zoo.data) con 101 transacciones (5 Kb), 18 atributos y con información de 101 animales y se duplicaron hasta obtener un máximo de 40.400 transacciones.

Para evaluar el rendimiento de la primitiva *Mate by*, se utilizó la herramienta *Mate-KDD* , un clasificador medianamente acoplado con el SGBD PostgreSQL, una herramienta desarrollada en el laboratorio de KDD de la Universidad de Nariño (Colombia), en la cual la primitiva *Mate by* se implemento como función definida por el usuario. El comportamiento de la primitiva *Mate by*, en las dos arquitecturas, cuando el número de atributos aumenta, se muestra en la figura 2.

Como se pude observar en la gráfica, el rendimiento de la primitiva Mate by es mejor en su versión fuertemente acoplada, cuando el número de atributos aumenta, debido al número de combinaciones que se deben realizar. Sin embargo las dos arquitecturas tienden a presentar un comportamiento exponencial en la medida que el número de atributos aumenta, de ahí que deben adoptar estrategias para disminuir el número de atributos a evaluar como seria un prepoda, evitando así combinaciones innecesarias.

8. Conclusiones

Se aplicó el método tres-pasos [29][30][31] para integrar fuertemente la tarea de clasificación al interior del SGBD PostgreSQL[24] [18]. Se extendió el álgebra relacional con los nuevos operadores *Mate*,

Entro, *Gain* y *Describe Classifier*. Se extendió el lenguaje SQL con la primitiva *Mate by*, las funciones agregadas *Gain()* y *Entro()* y el operador SQL *Describe Classification Rules.*

De esta manera se complemento el proyecto PostgresKDD, con trabajos anteriores sobre Asociación [29][32]. El resultado es un sistema DCBD fuertemente acoplado con PostgreSQL para soportar el descubrimiento de reglas tanto de Clasificación, como de Asociación.

Como trabajos futuros están el de integrar fuertemente las funciones agregadas *Entro()*, *Gain()* y el operador *Describe Classifier Rules* que actualmente están medianamente acoplados y realizar pruebas de las primitivas de Clasificación con conjuntos de datos reales y evaluar su desempeño con respecto a otros algoritmos de clasificación por árboles de decisión.

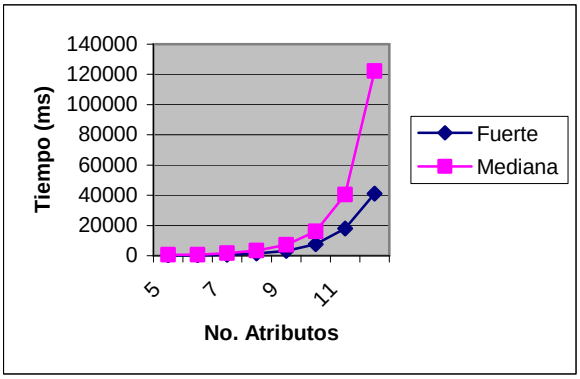


Figura 2. Comportamiento primitiva Mate by

Referencias

[1] Agrawal R., Mehta M., Shafer J., Srikant R., Arning A., Bollinger T., The Quest Data Mining System, 2º Conference KDD y Data Mining, Portland, Oregon, 1996.

[2] Boulicaut J-F., Masson C., Data Mining Query Languages, Data Mining and Knowledge Discovery Handbook : A Complete Guide for Practitioners and Researchers, O. Maimon and L. Rokach (Eds), Kluwer Academic Publishers, 14 pages, 2005.

[3] Chaudhuri S., Data Mining and Database Systems: Where is the Intersection?, Bulletin of the Technical Committee on Data Engineering, Vol. 21 No. 1, Marzo, 1998.

[4] Clear, J., Dunn, D., Harvey, B., Heytens, M., Lohman, P., Mehta, A., Melton, M., Rohrberg, L., Savasere, A., Wehrmeister, R., Xu, M., NonStop SQL/MX Primitives for Knowledge Discovery, KDD-99, San Diego, USA, 1999.

[5] Clear, J., Dunn, D., Harvey, B., Heytens, M., Lohman, P., Mehta, A., Melton, M., Rohrberg, L., Savasere, A., Wehrmeister, R., Xu, M., Large Scale Knowledge Discovery Using NonStop SQL/MX, Technical Report, Compaq Computer Corporation, Tandem Division, 1999.

[6] Freitas, A.A., Generic, Set-oriented Primitives to Support Data-parallel Knowledge Discovery in Relational Database Systems, Ph.D. diss., http://cswww.essex.ac.uk/SystemsArchitecture/DataMining/alex/thesis.html, 1997.

[7] Freitas, A.A., Lavington, S.H., Using SQL Primitives and Parallel DBServers to Speed Up Knowledge Discovery in large relational databases, University of Essex, UK, http:// citeseer.nj.nec.com/cs, 1997.

[8] Han J., Fu Y., Wang W., Chiang J., Zaiane O., Koperski K., DBMiner: Interactive Mining of Multiple-Level Knowledge in Relational Databases, ACM SIGMOD, Montreal, Canada, 1996.

[9] Han, J., Fu Y., Wang W., Koperski K., Zaiane O., DMQL: A Data Mining Query Language for Relational Databases, SIGMOD 96 Workshop, On research issues on Data Mining and Knowledge Discovery DMKD 96, Montreal, Canada, 1996.

[10] Imielinski, T., Mannila, H., A Database Perspective on Knowledge Discovery, Communications of the ACM, Vol 39, No. 11, November, 1996.

[11] Imielinski T., Virmani A., MSQL: A Query Language for Database Mining, in journal Data Mining and Knowledge Discovery, Kluwer Academica Publishers, Volume 3, Number 4, 1999.

[12] ISO/IEC 13249-1:2000, Information technology – Database languages – SQL Multimedia and Application Packages – Part 1: Framework, International Organization for Standardization, 2000.

[13] ISO/IEC 13249-6:2002, Information technology – Database languages – SQL Multimedia and Application Packages – Part 6: Data Mining, International Organization for Standardization, 2002.

[14] Metha M., Agrawal R., Rissanen J., SLIQ: A Fast Scalable Classifier for Data Mining, Proceedings EDBT96, Avignon, France, 1996.

[15] Melton, J., Eisenberg, A., SQL Multimedia and Application Packages (SQL/MM), in http://www.sigmod.org/records/issues/0112/standars.pdf, 2001.

[16] Meo R., Psaila G., Ceri S., An Extension to SQL for Mining Association Rules, Data Mining and Knowledge Discovery, Kluwer Academic Publishers, Vol 2, pp .195-224, Boston, 1998.

[17] Microsoft Corporation, OLE DB for Data Mining Draft Specification, version 0.9, in http://www.microsoft.com/data/oledb/dm.html, 2000.

[18] Momjian, B., PostgreSQL: Introduction and Concepts, Addison-Wesley, May 4, 2000.

[19] Netz A., Chaudhuri S., Bernhardt J., Fayyad U., Integration of Data Mining and Relational Databases, Proceedings of the 26th International Conference on Very Large Databases, Cairo, Egytpr,2000.

[20] Netz A., Chaudhuri S., Fayyad U.,Bernhardt J., Integrating Data Mining with SQL Databases: OLE DB for Data Mining, technical report,2000.

[21] Quinlan J.R., Induction of decision trees. Machine Learning, 81-106, 1986.

[22] Quinlan J.R., C4.5: Programs for Machine Leraning, Morgan Kaufmann Publishers, 1993.

[23] Schwenkreis, F., New working draft of SQL/MM Part 6: Data Mining based on BHX008 and BHX033-BHX039, ISO/IEC JTC1/SC32 Data Management and Interchange WG$ SQL/MM, Secretariat USA (ANSI), may, 2000.

[24] Stonebraker, M., Rowe, L., A., The Design of POSTGRES, Proceedings of the ACM-SIGMOD Conference, Washington D.C., 1986.

[25] Sattler K, Dunemann O., SQL Database Primitives for Decision Tree Classifiers, CIKM, Atlanta,Georgia, USA, November, 2001.

[26] Shafer J., Agrawal R., Metha M., SPRINT: A Scalable Parallel Classifier for Data Mining, Proceedings of the VLDB Conference, Bombay, India,1996.

[27] Sarawagi S., Thomas S., Agrawal R., Integrating Association Rule Mining with Relational Database Systems: Alternatives and Implications, Data Mining and Knowledge Discovery, Kluwer Academic Publishers, Vol 4, 2000.

[28] Timarán, R., Arquitecturas de Integración del Proceso de Descubrimiento de Conocimiento con Sistemas de Gestión de bases de datos: un Estado del Arte, en revista Ingeniería y Competitividad, Universidad del Valle, Volumen 3, No. 2, Cali, diciembre de 2001.

[29] Timarán R., Millán M., Machuca F., New AlgebraicOperators and SQL Primitives for Mining Association Rules, Proceedings of the IASTED International Conference Neural Networks and Computational Intelligence, Cancun, Mexico, 2003.

[30] Timarán, R., Millán, M., EquipAsso: un algoritmo para el descubrimiento de Reglas de Asociación basado en operadores algebraicos, en memorias de la 4ª Conferencia Iberoamericana en Sistemas, Cibernética e Informática CICI 2005, Orlando, Florida, EE.UU., julio 2005.

[31] Timarán, R., Millán, M., EquipAsso: an Algorithm based on New Relational Algebraic Operators for Association Rules Discovery, in proceedings of the Fourth IASTED International Conference on Computational Intelligence, Calgary, Alberta, Canada, july 2005.

[32] Timarán, R., Guerrero,M., Diaz, M., Cerquera, C., Armero,S., Implementación de Primitivas SQL para Reglas de Asociación en una Arquitectura Fuertemente Acoplada, en memorias de la XXXI Conferencia Latinoamericana de Informática CLEI 2005, Santiago de Cali, Colombia, octubre 2005.

[33] Thomas, S., Sarawgi, S., Mining Generalized Association Rules and Sequential Patterns Using SQL Queries, in Fourth International Conference on Knowledge Discovery and Data Mining, KDD, 1998.

[34] Yoshizawa, T., Pramudiono, I., Kitsuregawa, M, SQL Based Association Rule Mining using Commercial RDBMS (IBM DB2 UDB EEE), Data Warehousing and Knowledge Discovery, pag. 301-306, 2000.

[35] Wang, M., Iyer, B., Scott, V.,J., Scalable Mining for Classification Rules in Relational Databases, International Database Engineering and Application Symposium, pages 58-67, 1998.

Certificación de propiedades usando distintos probadores de teoremas: un caso de estudio*

J. Santiago Jorge Víctor M. Gulías
Laura M. Castro
MADS Group, Dept. de Computación, Universidade da Coruña
Campus de Elviña, s/n., 15071 A Coruña, España
{sjorge, gulias, lcastro}@udc.es

Abstract

Exploramos el uso de probadores de teoremas para certificar propiedades del software. En concreto, se utilizan dos asistentes de prueba diferentes para ilustrar el método: COQ *y* PVS. *Comparando dos probadores de teoremas podemos llegar a conclusiones sobre su conveniencia. Aplicamos las técnicas en un escenario del mundo real, un modelo de un algoritmo de asignación de bloques de memoria para un servidor de vídeo bajo demanda. El desarrollo consta de dos pasos: primero, la definición en los asistentes de pruebas* COQ *y* PVS *del algoritmo y ciertas propiedades relevantes a probar; y después, la codificación de los teoremas.*

1. Introducción

La comprobación de la corrección de un programa, incluso de los más simples, es una labor compleja, y al mismo tiempo, se trata de una de las actividades más importantes en el desarrollo del software. Los métodos formales [3, 14] para la prueba de la corrección de un programa tienen por objeto incrementar nuestra comprensión de los programas y proporcionar confianza en el software ya que están basados en representaciones y razonamientos matemáticos. Se destacan dos planteamientos: *model checking* cuando se prueba que *toda* ejecución posible satisface la especificación, y *prueba de teoremas* en el que se construye una prueba matemática de que el software cumple su especificación. En el presente trabajo exploramos esta segunda aproximación.

El uso de probadores de teoremas ayuda al verificador en sus desarrollos. Estas herramientas proporcionan control y asistencia durante los procesos de especificación y prueba; por tanto evitan errores que podrían introducirse en una prueba manual. Los probadores de teoremas están basados en alguna lógica (lógica proposicional, lógica de primer orden, teoría de tipos constructiva, etc.) que provee un marco para la formalización matemática. Aunque algunas partes de la verificación son automatizables, se trata, en principio, de una técnica manual asistida por computador. En los asistentes de prueba interactivos el usuario es guiado de forma interactiva por el sistema en el proceso de construcción de una prueba, siendo verificada inmediatamente la validez de cada paso del desarrollo.

En este trabajo exponemos la verificación formal de una parte de la funcionalidad de una aplicación del mundo real: la implementación de un servidor distribuido de vídeo bajo demanda [7] desarrollado con el lenguaje funcional concurrente ERLANG.La programación funcional [2, 9] permite verificar la corrección de programas de una manera más sencilla y, gracias a la transparencia referencial, es posible establecer y probar distintas propiedades sobre funciones y utilizar éstas para transformar programas en otros equivalentes.

El artículo es una continuación de los trabajos en [11, 12] donde verificamos en el asistente de pruebas COQ un algoritmo de asignación de bloques de memoria para un servidor de vídeo bajo demanda. En el presente estudio usamos dos probadores de teoremas: COQ y PVS. Comparando dos asistentes de prueba, podemos alcanzar conclusiones generales sobre su idoneidad. El sistema COQ es una implementación del *cálculo de construcciones inductivas* [4, 13], desarrollado en el INRIA. Este sistema es un probador automático de teoremas dirigido a objetivos por medio de tácticas en el estilo del LCF. Por otro lado, PVS es un sistema de verificación desarrollado por SRI a lo largo de casi veinte años de experiencia. Integra un lenguaje de especificación expresivo y potentes capacidades de prueba de teoremas. PVS ha sido usado con éxito en proyectos extensos y complejos tanto en el ámbito académico como en el industrial. De todos modos, la elección de estos probadores no es crucial ya que

*Parcialmente financiado por MEC TIN2005-08986 y por Xunta de Galicia PGIDIT05TIC10503PR

las pruebas presentadas son fácilmente construibles en otros asistentes similares.

Este trabajo está estructurado de la siguiente forma: primero describimos brevemente un algoritmo de asignación de bloques de memoria para un servidor de vídeo bajo demanda. A continuación, mostramos los modelos PVS y COQ usados en la descripción del algoritmo. En las secciones 6, 7 y 8 se certifica la corrección de un modelo del programa con ambos probadores. Por último, presentamos las conclusiones.

2. El proyecto VoDKA

Un servidor de vídeo bajo demanda es un sistema que proporciona servicio de vídeo a muchos clientes que realizan solicitudes en cualquier momento, i.e. sin limitaciones temporales preestablecidas. Estos servidores son complejos sistemas distribuidos que tienen que satisfacer algunos requisitos críticos: gran capacidad de almacenamiento, manejo de miles de transacciones concurrentes garantizando tiempos de respuesta razonables e idealmente predecibles, gran ancho de banda, fiabilidad, escalabilidad, adaptabilidad a la topología subyacente, y bajo coste.

El objetivo del proyecto VoDKA (*Video-on-Demand Kernel Architecture*, `http://madsgroup.org/vodka`) es la construcción de un servidor escalable, tolerante a fallos, multiprotocolo y adaptable (tanto a la topología de la red como a los protocolos de los usuarios finales). Se proponen clusters de ordenadores de bajo coste como una solución asequible para la gran cantidad de recursos requeridas. Sin embargo, el problema principal es cómo diseñar e implementar una aplicación distribuida para controlar dicho cluster satisfaciendo los requisitos de un sistema de vídeo bajo demanda (escalabilidad, adaptabilidad, disponibilidad, flexibilidad).

En el primer enfoque del diseño, los componentes del cluster se organizan como un sistema de almacenamiento jerárquico, identificándose tres niveles especializados: (a) *Primario o Repositorio*, cuyo objetivo es el almacenamiento de todos los objetos multimedia; (b) *Secundario o Caché*, a cargo del almacenamiento de los vídeos leídos del repositorio antes de su envío al cliente; y (c) *Terciario o Streaming*, encargado de la adaptación de protocolos y del envío de los objetos multimedia en el formato apropiado al cliente.

3. El problema estudiado: la asignación de bloques en la caché

El nivel de *caché* proporciona un alto *throughput* (i.e. la cantidad de datos enviados en un período de tiempo dado) para aliviar las habituales deficiencias (en términos de ancho de banda y concurrencia) del repositorio. Este nivel está compuesto por un conjunto de nodos donde se ejecutan algunos algoritmos complejos y susceptibles de contener errores. La porción de software que va a ser objeto de verificación en este trabajo es parte de un subsistema de la caché: el algoritmo de asignación de bloques. Se han aplicado también métodos formales a otras partes [1, 8].

El disco local de cada nodo del nivel secundario actúa como una *caché* local dividida en bloques de tamaño fijo (e.g. 8 MB) que almacenan fragmentos de un objeto multimedia. Al atender una petición, se busca primero el objeto multimedia en las cachés locales, evitando el acceso al nivel terciario si fuese posible.

Si delegamos la petición del objeto multimedia al nivel terciario, habrá que reservar el espacio (garantizándolo) en la caché local de algún nodo del nivel secundario. Si para ello hubiese que liberar unos bloques, hay que garantizar que no estén siendo usados. Por motivos de rendimiento, resulta además importante que los bloques de un mismo fichero estén lo más cerca posible, luego las asociaciones de vectores en caché serán lo más secuenciales posibles. Por ello hablaremos de *intervalos de bloques*.

4. Representación de los intervalos de bloques

Representamos cada intervalo como un triple (a, b, x) denotando que el intervalo comprendido entre los bloques a y b, ambos incluidos, tiene x clientes pendientes. Guardamos por tanto la secuencia de intervalos de bloques sobre los que existen peticiones, como una lista de intervalos como sigue:

$$[(a_1, b_1, x_1), (a_2, b_2, x_2), \ldots, (a_n, b_n, x_n)]$$

Por motivo de eficiencia, mantenemos los intervalos *ordenados* $\forall i, a_i \leq b_i \wedge b_i < a_{i+1}$. Y para ahorrar espacio, mantenemos la secuencia *compactada*, i.e. $\forall i, x_i \neq 0$ y $(x_i \neq x_{i+1}) \vee ((x_i = x_{i+1}) \wedge (b_i + 1 < a_{i+1}))$

Como vamos a reescribir el código fuente orignal en algún modelo equivalente, necesitaremos algunas definiciones previas. Mostramos tanto la aproximación con COQ como con PVS destacando las diferencias entre ambos probadores. Recúrrase a [6, 10] para una introducción a COQ; y un tutorial sobre PVS está disponible en [5]. En el texto proporcionamos el código de los ejemplos y la entrada de datos del usuario con la familia de tipo de letra `typewriter`, y la respuesta del sistema con la misma familia en *`itálica`*.

Coq Los intervalos de bloques están representados por tripletas de números naturales: las posiciones de comienzo y fin (de los bloques de memoria) y el número de peticiones pendientes existentes sobre ellos.

```
Inductive interval: Set:= tuple: nat->nat->nat->interval.
```

Se trata de una definición inductiva con un constructor, `tuple`, que recibe tres números naturales para crear un intervalo. Representamos una secuencia de intervalos como una lista de bloques usando una definición inductiva. El constructor `empty` representa una secuencia vacía mientras `make` toma un intervalo y una secuencia para formar otra secuencia.

```
Inductive seq: Set:= empty: seq
                   | make: interval->seq->seq.
```

Mantenemos los intervalos ordenados, i.e. para toda secuencia $[(a_1, b_1, x_1), \ldots, (a_n, b_n, x_n)]$, se cumplirá que $\forall i, a_i \leq b_i \wedge b_i < a_{i+1}$.

```
Fixpoint add (n:nat) (l: seq) {struct l}: seq:=
 match l with
 | empty => make (tuple n n (S O)) empty
 | make (tuple a b x) l' =>
   match le_lt_dec a n with
   | left _ =>
     match le_lt_dec n b with
     | left _ =>
       match eq_nat_dec a n with
       | left _ =>
         match eq_nat_dec n b with
         | left _ => make (tuple a b (S x)) l'
         | right _ =>
           make (tuple a a (S x))
             (make (tuple (S a) b x) l')
         end
       | right _ =>
         match eq_nat_dec n b with
         | left _ =>
           make (tuple a (pred b) x)
             (make (tuple b b (S x)) l')
         | right _ =>
           make (tuple a (pred n) x)
             (make (tuple n n (S x))
               (make (tuple (S n) b x) l'))
         end
       end
     | right _ => make (tuple a b x) (add n l')
     end
   | right _ => make (tuple n n (S O)) l
   end
 end.
```

Figura 1. Parte del modelo COQ

PVS Las definiciones son análogas a las utilizadas en el modelo COQ: un intervalo es una tupla de tres elementos, y una secuencia es una lista de intervalos.

```
interval: TYPE = [nat, nat, nat]
seq: TYPE = list[interval]
```

5. El algoritmo: Asignación de bloques

El núcleo del algoritmo es la definición `add` que añade peticiones sobre bloques de memoria. La simulación del sistema real sobre casos de prueba representativos reveló algunos errores, pero desafortunadamente no garantizaba la corrección del software. Para certificar la corrección del programa, lo reescribimos en algún modelo equivalente al código fuente original. El inconveniente de este método es que no estamos verificando el sistema real sino un *modelo* abstracto del mismo; sin embargo, este proceso incrementa la fiabilidad del sistema (o al menos la confianza en el mismo) [14].

```
add(n: nat, l: seq): RECURSIVE seq =
 CASES l
  OF null: make(tuple(n, n, 1), null),
     cons(i, l1):
      IF i'1 <= n
      THEN
        IF n <= i'2
        THEN
          IF i'1 = n
          THEN
            IF n = i'2
            THEN make(tuple(i'1, i'2, i'3 + 1), l1)
            ELSE make(tuple(i'1, i'1, i'3 + 1),
                   make(tuple(i'1 + 1, i'2, i'3), l1))
            ENDIF
          ELSE
            IF n = i'2
            THEN make(tuple(i'1, i'2 - 1, i'3),
                   make(tuple(i'2, i'2, i'3 + 1), l1))
            ELSE make(tuple(i'1, n - 1, i'3),
                   make(tuple(n, n, i'3 + 1),
                     make(tuple(n + 1, i'2, i'3), l1)))
            ENDIF
          ENDIF
        ELSE make(i, add(n, l1))
        ENDIF
      ELSE make(tuple(n, n, 1), l)
      ENDIF
  ENDCASES
 MEASURE length(l)
```

Figura 2. Parte del modelo PVS

La figura 1 muestra parte del modelo COQ. La definición de `add` se realiza por recursión estructural en el segundo argumento: la secuencia de intervalos. Esto se denota mediante `{struct l}`. Para su aceptación por COQ, `add` satisfará una condición sintáctica por la cual el argumento de terminación, `l`, tiene que decrecer estructuralmente en cada llamada recursiva. El cuerpo de la función `add` presenta diversas alternativas basadas en análisis de casos. El motivo de la especial complicación de esta definición es el mantenimiento de la secuencia en su orden ascendente, lo cual implica que bajo algunas condiciones habrá que partir un intervalo. La figura 2 muestra la misma traducción a PVS. La cláusula `MEASURE` se especifica para garantizar que la definición es total indicando el argumento que decrece en cada llamada recursiva. PVS usa esta cláusula para la generación de obligaciones, *Type-Correctness Condition* (TCC), que aseguren que la función siempre termina.

Como vemos, COQ y PVS poseen distintas soluciones para la definición de funciones recursivas primitivas. COQ es más restrictivo porque exige el decrecimiento estructural de uno de los argumentos de la función. La solución de PVS resulta más flexible; las obligaciones generadas por PVS en la definición de `add` para garantizar su terminación son las

siguientes:

```
add_TCC1: OBLIGATION
  FORALL (i: interval,l1: list[interval],l: seq,n: nat):
    l = cons(i, l1) AND i‘1 <= n AND n <= i‘2 AND
    NOT i‘1 = n AND n = i‘2 IMPLIES i‘2 - 1 >= 0;

add_TCC2: OBLIGATION
  FORALL (i: interval,l1: list[interval],l: seq,n: nat):
    l = cons(i, l1) AND i‘1 <= n AND n <= i‘2 AND
    NOT i‘1 = n AND NOT n = i‘2 IMPLIES n - 1 >= 0;
```

Las dos obligaciones anteriores son probadas automáticamente por la estrategia estándar de PVS para la demostración de TCCs. Las restantes obligaciones generadas por PVS en el modelo que estamos exponiendo en este trabajo, también son probadas de forma automática por el sistema y no las vamos a mostrar.

6. Prueba de una propiedad de `add` en COQ

Primero necesitamos una función auxiliar `nth` que nos devuelva el número de peticiones existentes sobre el n-ésimo bloque de memoria. Usaremos esta función, certificada en [11], para formular la especificación.

```
Fixpoint nth (n:nat) (l:seq) {struct l}: nat :=
  match l with
    | empty => O
    | make (tuple a b x) l' =>
        match le_lt_dec a n with
          | left _ => match le_lt_dec n b with
                        | left _ => x
                        | right _ => nth n l'
                      end
          | right _ => O
        end
  end.
```

La función `add` tiene que cumplir que tras su ejecución, el número de peticiones sobre el bloque al que se le añadió otra nueva habrá sido incrementado mientras que todos los demás bloques permanecerán con el mismo número de peticiones que tenían antes de esta operación.

$$\texttt{nth n (add i l)} = \begin{cases} \texttt{nth n l} & \text{if } \texttt{i} \neq \texttt{n} \\ 1 + \texttt{nth n l} & \text{if } \texttt{i} = \texttt{n} \end{cases}$$

Subdividimos esta propiedad en dos teoremas según `i` sea igual a `n` o no.

```
Theorem nth_add_1 : forall (l:seq) (n m:nat),
    n=m -> nth n (add m l) = S (nth n l).
```

La prueba de una propiedad se desarrolla aplicando sucesivas *tácticas* que resuelven la conclusión o la transforman en nuevos objetivos. La prueba terminará cuando no quede ninguna meta por probar. Tanto la definición de `add` como la de `nth` se realizan por recursión estructural en la secuencia de intervalos de bloques, luego procedemos por inducción en `l`. La táctica `induction` permite el uso del esquema de inducción asociado a cualquier definición inductiva; con el parámetro `as` indicamos los nombres de las variables que vamos a introducir en el contexto.

```
induction l as [|i l' indh].
(* base case *)
intros n m h.
rewrite h.
simpl.
```

La táctica `intros` mueve los cuantificadores universales y las premisas de las implicaciones del objetivo de la prueba al entorno de las hipótesis. Con `rewrite` aplicamos razonamiento ecuacional, sustituyendo “iguales por iguales”. El uso de la táctica `simpl` lleva a cabo una simplificación realizando todas las posibles β-reducciones. Al desarrollar las pruebas, las premisas aparecen sobre una doble línea y la meta a probar, debajo:

```
2 subgoals

  n : nat
  m : nat
  h : n = m
  ============================
   (if le_lt_dec m m then
      if le_lt_dec m m then 1 else 0 else 0) = 1

subgoal 2 is:
 forall n m : nat,
  n=m -> nth n (add m (make i l'))=S (nth n (make i l'))
```

El análisis de casos aplicado sobre `le_lt_dec n n` devuelve o bien el valor `(left _)` con una prueba de `le n n`, o el valor `(right _)` con una prueba de `lt n n`. Como todo número natural es igual a sí mismo, la segunda alternativa nos llevaría a una hipótesis errónea $n < n$ y se resolvería de forma inmediata. La abstracción de este patrón de verificación común a diferentes pruebas nos sugiere la creación de una nueva táctica *ad hoc*, `lelt`, que nos va a permitir eliminar algunos pasos repetitivos en el futuro.

```
Tactic Definition DecLeLt a b:=
    Case (le_lt_dec a b); Auto
    Orelse (Intros; Try Contradict).
```

Aplicamos ahora `lelt`:

```
lelt m m.

1 subgoal

  i : interval
  l' : seq
  indh : forall n m : nat,
         n=m -> nth n (add m l')=S (nth n l')
  ============================
   forall n m : nat,
   n=m -> nth n (add m (make i l'))=S (nth n (make i l'))
```

En el paso inductivo comenzamos introduciendo al entorno de las hipótesis los cuantificadores universales y la hipótesis del teorema, reescribimos esta hipótesis tras descomponer con la táctica *ad hoc* `casei` el intervalo `i` en el triplete de número naturales que lo forman, y realizamos todas las β-reducciones posibles.

```
(* ind. hyp. *)
intros n m h.
casei i a b x.
rewrite h.
clear n h.
simpl.
```

```
1 subgoal

  l' : seq
  indh : forall n m : nat,
         n = m -> nth n (add m l') = S (nth n l')
  m : nat
  a : nat
  b : nat
  x : nat
  ============================
   nth m
     (if le_lt_dec a m
      then
       if le_lt_dec m b
       then
        if eq_nat_dec a m
        then
         if eq_nat_dec m b
         then make (tuple a b (S x)) l'
         else make (tuple a a (S x))
                   (make (tuple (S a) b x) l')
        else
         if eq_nat_dec m b
         then make (tuple a (pred b) x)
                   (make (tuple b b (S x)) l')
         else make (tuple a (pred m) x)
               (make (tuple m m (S x))
                     (make (tuple (S m) b x) l'))
       else make (tuple a b x) (add m l')
      else make (tuple m m 1) (make (tuple a b x) l')) =
    S (if le_lt_dec a m then
       if le_lt_dec m b then x else nth m l' else 0)
```

Siguiendo la estructura de la función `add` efectuamos los siguientes tratamientos por casos seguidos de la reescritura de las igualdades de las hipótesis y la realización de todas las β-reducciones posibles.

```
case (le_lt_dec a m); intro hle1;
  case (le_lt_dec m b); intro hle2;
    case (eq_nat_dec a m); intro heq1;
      case (eq_nat_dec m b); intro heq2;
        try rewrite heq1; try rewrite heq2; simpl.

16 subgoals

  l' : seq
  indh : forall n m : nat,
         n = m -> nth n (add m l') = S (nth n l')
  m : nat
  a : nat
  b : nat
  x : nat
  hle1 : a <= m
  hle2 : m <= b
  heq1 : a = m
  heq2 : m = b
  ============================
   (if le_lt_dec b b then if le_lt_dec b b then S x
    else nth b l' else 0) = S x

subgoal 2 is:...
```

Obtenemos dieciséis alternativas. Todas son resueltas por la táctica *ad hoc* `lelt` salvo dos en las que también aplicamos la hipótesis de inducción.

```
(* a<=m, m<=b, a=m,  m=b  *) lelt b b.
(* a<=m, m<=b, a=m,  m<>b *) lelt m m.
(* a<=m, m<=b, a<>m, m=b  *) lelt a b. lelt b (pred b).
                             lelt b b.
(* a<=m, m<=b, a<>m, m<>b *) lelt a m. lelt m (pred m).
                             lelt m m.
(* a<=m, m>b,  a=m,  m=b  *) lelt m m.
(* a<=m, m>b,  a=m,  m<>b *) lelt m m. lelt m b.
                             apply indh. auto.
(* a<=m, m>b,  a<>m, m=b  *) lelt a b.
(* a<=m, m>b,  a<>m, m<>b *) lelt a m. lelt m b.
                             apply indh. auto.
(* a>m,  m<=b, a=m,  m=b  *) lelt b b.
(* a>m,  m<=b, a=m,  m<>b *) lelt m m.
(* a>m,  m<=b, a<>m, m=b  *) lelt b b.
(* a>m,  m<=b, a<>m, m<>b *) lelt m m.
(* a>m,  m>b,  a=m,  m=b  *) lelt b b.
(* a>m,  m>b,  a=m,  m<>b *) lelt m m.
(* a>m,  m>b,  a<>m, m=b  *) lelt b b.
(* a>m,  m>b,  a<>m, m<>b *) lelt m m.
Qed.
```

Con esto hemos demostrado ya el primero de los dos teoremas de la especificación del algoritmo `add`. El segundo teorema se plantea del siguiente modo:

```
Theorem nth_add_2 : forall (l:seq) (n m:nat),
  n<>m -> nth n (add m l) = nth n l.
```

La demostración es similar a la del teorema anterior.

7. Prueba de una propiedad de `add` en PVS

Tal como hicimos en COQ, definimos una función auxiliar `nth` para formular la especificación.

```
nth(n: nat, l: seq, default: nat): RECURSIVE nat =
  CASES l
    OF null: default,
       cons(i, l1):
         IF i‘1 <= n
           THEN IF n <= i‘2 THEN i‘3
                ELSE nth(n, l1, default) ENDIF
         ELSE default
         ENDIF
    ENDCASES
   MEASURE length(l)
```

El correspondiente primer teorema se formula del modo siguiente:

```
nth_add_1: THEOREM FORALL (m, n: nat), (l: seq):
  m = n => nth(n, add(m, l), 0) = 1 + nth(n, l, 0)
```

La prueba del teorema en PVS es resuelta por el sistema automáticamente mediante la táctica avanzada `induct-and-simplify`. Resulta aconsejable usar en primer lugar los comandos más potentes, dejando los más simples para cuando los anteriores no tengan éxito. En comparación a COQ, el asistente PVS automatiza una gran cantidad de trabajo, aliviando el trabajo del ingeniero.

```
nth_add_1 :
  |-------
1   FORALL (m, n: nat), (l: seq):
        m = n => nth(n, add(m, l), 0) = 1 + nth(n, l, 0)

(induct-and-simplify "l")

By induction on l, and by repeatedly rewriting and
simplifying, Q.E.D.
```

El segundo teorema se resuelve del mismo modo.

```
nth_add_2: THEOREM FORALL (m, n: nat), (l: seq):
  m /= n => nth(n, add(m, l), 0) = nth(n, l, 0)
```

```
Inductive ascend : seq->Prop :=
  | ascend1 : ascend empty
  | ascend2 : forall a b x:nat,
      le a b -> ascend (make (tuple a b x) empty)
  | ascend3 : forall (l:seq) (a b x c d y:nat),
      le a b -> lt b c ->
        ascend (make (tuple c d y) l) ->
          ascend (make (tuple a b x)
                    (make (tuple c d y) l)).

Inductive packed : seq->Prop :=
  | packed1 : packed empty
  | packed2 : forall a b x:nat,
      le a b -> packed (make (tuple a b x) empty)
  | packed3 : forall (l:seq) (a b x c d y:nat),
      packed (make (tuple c d y) l) ->
        le a b -> x<>y -> lt b c ->
          packed (make (tuple a b x)
                    (make (tuple c d y) l))
  | packed4 : forall (l:seq) (a b x c d y:nat),
      packed (make (tuple c d y) l) ->
        le a b -> lt (S b) c ->
          packed (make (tuple a b x)
                    (make (tuple c d y) l)).
```

```
ascend(l: seq): RECURSIVE bool =
  CASES l
    OF null: TRUE,
       cons(x, xs):
         CASES xs
           OF null: x‘1 <= x‘2,
              cons(y, ys): ascend(xs) AND
                             x‘1 <= x‘2 AND x‘2 < y‘1
           ENDCASES
    ENDCASES
   MEASURE length(l)

packed(l: seq): RECURSIVE bool =
  CASES l
    OF null: TRUE,
       cons(x, xs):
         CASES xs
           OF null: x‘1 <= x‘2,
              cons(y, ys):
                packed(xs) AND x‘1 <= x‘2 AND
                  ((x‘3 /= y‘3 AND x‘2 < y‘1) OR
                   (x‘2 + 1) < y‘1)
           ENDCASES
    ENDCASES
   MEASURE length(l)
```

Figura 3. Definiciones `ascend` **y** `packed` **en Coq (izq.) y Pvs (der.)**

8. Forma canónica de las secuencias

Las secuencias de intervalos estarán representadas en todo momento siguiendo una “forma canónica o normal”, de modo que dos representaciones diferentes siempre se corresponderán con dos objetos distintos. El uso de una forma canónica nos va a proporcionar un mejor comportamiento espacial y temporal. De cara a especificar esta forma canónica introducimos una serie de definiciones, y a continuación demostraremos que la aplicación de la función `add` sobre una secuencia de intervalos de bloques en forma canónica nos proporciona un resultado que también está en forma canónica. El predicado `ascend` (figura 3) nos indica si una secuencia $[(a_1, b_1, x_1), \ldots, (a_n, b_n, x_n)]$ está en orden ascendente, es decir, nos indica si $\forall i, a_i \leq b_i \wedge b_i < a_{i+1}$ es cierto o no.

El predicado `ascend` no nos llega para definir la forma canónica ya que una misma secuencia de intervalos podría representarse de varias formas. Por ejemplo, la secuencia `[(2,2,3),(3,6,3),(8,9,2)]` también se podría representar por `[(2,6,3),(8,9,2)]`, así mismo en orden ascendente y de forma más compacta. Tenemos que exigir que la representación, además de ascendente, esté *compacta*, i.e. tiene que cumplirse que $\forall i, a_i \leq b_i \wedge ((x_i \neq x_{i+1} \wedge b_i < a_{i+1}) \vee (x_i = x_{i+1} \wedge b_i + 1 < a_{i+1}))$. La definición `packed` refleja el cumplimiento de las condiciones anteriores.

El resultado devuelto por `add` va a cumplir el predicado `ascend`, pero no necesariamente el predicado `packed` y por ello necesitamos definir una nueva función, `pack`, que a partir de una secuencia de intervalos de bloques con la propiedad ascendente, construya otra equivalente que además esté compactada (figura 4). En la definición de `pack` usamos la función auxiliar `packAux`.

Para completar la forma canónica necesitamos un predicado, `strict_positive`, que nos asegure que no hay en la secuencia de intervalos ningún bloque de memoria sin peticiones (figura 5).

Demostramos que la función `pack` devuelve una secuencia de intervalos equivalente a la que recibe como argumento siempre que esté en orden ascendente.

```
coq> Lemma nth_pack : forall (l:seq) (n:nat),
coq>  ascend l -> nth n l = nth n (pack l).

pvs> nth_pack: LEMMA FORALL (l: seq)(n: nat):
pvs>   ascend(l) => nth(n, l, 0) = nth(n, pack(l), 0)
```

La siguiente ley establece que el resultado de `add` es una secuencia de intervalos en orden ascendente.

```
coq> Lemma ascend_add : forall (l:seq) (n:nat),
coq>   ascend l -> ascend (add n l).

pvs> ascend_add: LEMMA FORALL (l: seq)(n: nat):
pvs>   ascend(l) => ascend(add(n, l))
```

A continuación demostramos que el resultado de la aplicación de `pack` sobre una secuencia ascendente es una secuencia compactada.

```
coq> Lemma packed_pack: forall (l:seq),
coq>   (ascend l) -> (packed (pack l)).

pvs> packed_pack: LEMMA FORALL (l: seq):
pvs>   ascend(l) => packed(pack(l))
```

También probamos que con `add` no dejamos bloques sin peticiones.

```
coq> Lemma strict_positive_add : forall (l:seq) (n:nat),
coq>   strict_positive l -> strict_positive (add n l).

pvs> strictPositive_add: LEMMA FORALL (l: seq)(n: nat):
pvs>   strictPositive(l) => strictPositive(add(n, l))
```

```
Fixpoint pack_aux (i:interval) (l:seq)
    {struct l} : seq:=
  match i with
  | tuple a b x =>
    match l with
    | empty => make i empty
    | make (tuple c d y) l' =>
      match eq_nat_dec x y with
      | left _ =>
        match eq_nat_dec (S b) c with
        | left _ =>
            pack_aux (tuple a d x) l'
        | right _ =>
            make i (pack_aux (tuple c d y) l')
        end
      | right _ =>
          make i (pack_aux (tuple c d y) l')
      end
    end
  end.

Definition pack (l:seq):seq := match l with
    | empty => empty
    | make i l' => pack_aux i l'
  end.
```

```
packAux(i: interval, l: seq): RECURSIVE seq =
  CASES l
    OF null: make(i, empty),
       cons(x, xs):
         IF i'3 = x'3
           THEN IF i'2 + 1 = x'1
                  THEN packAux(tuple(i'1, x'2, i'3), xs)
                ELSE make(i, packAux(x, xs))
                ENDIF
         ELSE make(i, packAux(x, xs))
         ENDIF
    ENDCASES
   MEASURE length(l)



pack(l: seq): seq =
    CASES l OF null: empty,
               cons(x, xs): packAux(x, xs)
      ENDCASES
```

Figura 4. Definiciones de `pack` y `packAux` en los modelos Coq (izq.) y Pvs (der.)

```
Inductive strict_positive: seq->Prop:=
  | strict_positive1: (strict_positive empty)
  | strict_positive2: forall (l:seq) (a b x: nat),
     strict_positive l -> lt O x
       -> strict_positive (make (tuple a b x) l).

Inductive canonical: seq->Prop:=
  canonical1: forall (l:seq),
    (packed l)->(strict_positive l)->(canonical l).
```

```
strictPositive (l: seq): RECURSIVE bool =
  CASES l OF null: TRUE,
            cons(x, xs): strictPositive(xs) AND x'3 > 0
    ENDCASES
   MEASURE length(l)

canonical(l: seq): bool = packed(l) AND
                            strictPositive(l)
```

Figura 5. Definiciones de `strictPositive` y `canonical` en los modelos Coq (izq.) y Pvs (der.)

Y lo mismo con la función `pack`.

```
coq> Lemma strict_positive_pack : forall (l:seq),
coq>   strict_positive l -> strict_positive (pack l).

pvs> strictPositive_pack: LEMMA FORALL (l: seq):
pvs>   strictPositive(l) => strictPositive(pack(l))
```

Con esto hemos demostrado que la función `pack` devuelve una secuencia de intervalos "equivalente" a la que recibe como argumento y que además está en forma canónica. Por último, la demostración de que dada una secuencia en forma canónica, añadiendo una petición sobre un bloque cualquiera con `add` y aplicando a continuación `pack` obtenemos una nueva secuencia en forma canónica (figura 6). En la prueba procedemos instanciando las variables, expandiendo la definición de `canonical` y resolviendo las submetas aplicando las leyes demostradas antes.

9. Conclusiones

Presentamos un caso de estudio que ilustra cómo los métodos formales, y en particular la prueba de teoremas, contribuyen a la producción de software certificado. La especificación y la verificación proporcionan un conocimiento profundo de un programa y sus tareas. Esto aumenta nuestra confianza en el sistema.

Las pruebas han sido desarrolladas con la asistencia de los probadores de teoremas COQ y PVS. Estos probadores tienen diferentes filosofías: COQ formaliza una lógica de alto orden con tipos inductivos, que proporciona una rigurosa noción de prueba. Por otro lado, PVS carece de esta rigurosa noción de prueba pero incorpora un potente procedimiento de automatización que reduce el trabajo a realizar por los ingenieros de software. En el modelo COQ hemos abstraído patrones de prueba comunes a distintas pruebas creando nuevas tácticas *ad hoc* que nos permiten eliminar pasos repetitivos en los procesos. Tal como ha podido observarse lo largo del estudio, en general, la prueba de algo elemental conlleva mucho esfuerzo, principalmente en el sistema COQ, menos automatizado que PVS. La identificación de subcasos y la verificación por separado de cada uno resulta siempre una estrategia útil.

El ejemplo estudiado representa parte de una aplicación del mundo real. Una prueba completa de un sistema puede consistir en cientos de teoremas que examinan cada componente o subsistema. A menudo es difícil, sino imposible, la aplicación de métodos formales a un sistema completo. Por tanto la tendencia es buscar métodos composicionales que

```
Theorem canonical_pack_add : forall (l:seq) (n:nat),
  (canonical l) -> (canonical (pack (add n l))).

intros l n h.
inversion_clear h as [l' hpack hpos].
constructor.

2 subgoals
  l : seq
  n : nat
  hpack : packed l
  hpos : strict_positive l
  ============================
   packed (pack (add n l))

subgoal 2 is:
 strict_positive (pack (add n l))

apply packed_pack.
apply ascend_add.
apply packed_ascend.
auto...
apply strict_positive_pack.
apply strict_positive_add.
auto...
Qed.
```

```
canonical_pack_add: THEOREM FORALL (l: seq) (n: nat):
  canonical(l) => canonical(pack(add(n, l)))

(skosimp*)
(expand "canonical")
(flatten)
(split)

this yields  2 subgoals:
canonical_pack_add.1 :
[-1]  packed(l!1)
[-2]  strictPositive(l!1)
  |-------
1   packed(pack(add(n!1, l!1)))

(use "packed_pack") (split -1)
(use "ascend_add") (split -1)
(use "packed_ascend") (split -1)

canonical_pack_add.2 :
[-1]  packed(l!1)
[-2]  strictPositive(l!1)
  |-------
1   strictPositive(pack(add(n!1, l!1)))

(use "strictPositive_pack") (split -1)
(use "strictPositive_add") (split -1)

Q.E.D.
```

Figura 6. Prueba del mantenimiento de la forma normal en los modelo Coq (izq.) y PVS (der.)

tratan de verificar o validar diferentes partes de una aplicación de forma separada, para luego llegar a conclusiones sobre el sistema al completo. Creemos que el futuro de la verificación de programas se encamina hacia la propuesta general de obtener bibliotecas de funciones o módulos certificados.

Los trabajos enfocados en simplificar o automatizar el uso de métodos formales como parte de las herramientas de los ingenieros de software conseguirán, con el paso del tiempo, reducir la distancia entre una especificación y un programa certificado que satisfaga dicha especificación.

Referencias

[1] T. Arts and J. J. Sánchez. Global scheduler properties derived from local restrictions. In *ACM Sigplan Erlang Workshop at the Principles, Logics, and Implementations of high-level programming languages (ACM Sigplan Erlang Workshop at PLI)*, pages 49–57. ACM Press, 2002.

[2] R. Bird and P. Wadler. *Introduction to Functional Programming*. Prentice Hall, Hertfordshire, UK, 1988.

[3] E. M. Clarke, J. M. Wing, R. Alur, R. Cleaveland, D. Dill, A. Emerson, S. Garland, S. German, J. Guttag, A. Hall, T. Henzinger, G. Holzmann, C. Jones, R. Kurshan, N. Leveson, K. McMillan, J. Moore, D. Peled, A. Pnueli, J. Rushby, N. Shankar, J. Sifakis, P. Sistla, B. Steffen, P. Wolper, J. Woodcock, and P. Zave. Formal methods: state of the art and future directions. *ACM Computing Surveys*, 28(4):626–643, 1996.

[4] T. Coquand and G. Huet. The calculus of constructions. *Information and Computation*, 76:95–120, 1988.

[5] J. Crow, S. Owre, J. Rushby, N. Shankar, and M. Srivas. A tutorial introduction to PVS. In *WIFT'95 Workshop on Industrial-Strength Formal Specification Techniques*, 1995.

[6] E. Giménez. A tutorial on recursive types in Coq. Technical report, INRIA, 1998. for Coq V6.2.

[7] V. M. Gulías, M. Barreiro, and J. L. Freire. VODKA: Developing a video-on-demand server using distributed functional programming. *Journal of Functional Programming*, 15(4):403–430, 2005.

[8] V. M. Gulías, A. Valderruten, and C. Abalde. Functional patterns for implementing distributed applications. In *IFIP/ACM Latin America Networking Conference (LANC'03)*, pages 89–98. ACM Press, 2003.

[9] P. Hudak. Conception, evolution, and application of functional programming languages. *ACM Computing Surveys*, 21(3):359–411, 1989.

[10] G. Huet, G. Kahn, and C. Paulin-Mohring. The Coq proof assistant: A tutorial, v8.0. Technical report, INRIA, 2004.

[11] J. S. Jorge. *Estudio de la verificación de propiedades de programas funcionales: de las pruebas manuales al uso de asistentes de pruebas*. PhD thesis, University of A Coruña, Spain, 2004.

[12] J. S. Jorge, V. M. Gulías, A. Valderruten, and D. Cabrero. Prueba de propiedades en la caché de un servidor funcional de vídeo bajo demanda. In *XXXI Conferencia Latinoamericana de Informática (CLEI 2005)*, 2005.

[13] C. Paulin-Mohring. Inductive definitions in the system Coq: Rules and properties. In M. Bezem and J. F. Groote, editors, *Typed Lambda Calculi and Applications (TLCA'93)*, volume 664 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993.

[14] D. A. Peled. *Software Reliability Methods*. Springer-Verlag, 2001.

GraspKM en la Recuperación de la Estructura de Software

Erick Vicente
Facultad de Ingeniería, Universidad Ricardo Palma
Lima 33, Perú
evicente@mail.urp.edu.pe

Manuel Tupia
Facultad de Ciencias e Ingeniería, Pontificia Universidad Católica del Perú
Lima 100, Perú
tupia.mf@pucp.edu.pe

Luis Rivera
Universidad Estadual Norte Fluminense, UENF, LCMAT-CCT
Campos dos Goytacazes, Rio de Janeiro, Brasil, 28015-620
rivera@uenf.br

Abstract

En la actualidad existe gran cantidad de sistemas de software carente de documentación, mas aún cuando se tratan de sistemas legados. En la literatura se han propuesto diversos métodos para obtener una abstracción de la estructura de estos sistemas. Estos métodos se encuentran basados principalmente en clustering, debido a los objetivos coincidentes de lo que se quiere de la estructura de un sistema y de la estructura de los clusters: los módulos de software deben ser altamente cohesivos y con bajo acoplamiento, de manera similar un cluster debe contener elementos que sean similares entre sí y que sean a su vez lo mas diferentes entre clusters. Los métodos encontrados en literatura para el clustering de software se encuentran clasificados dentro del clustering jerárquico. En el presente trabajo proponemos la adaptación del método KMeans en el contexto de GRASP, denominado GraspKM, para la búsqueda de la estructura de un sistema. Este método trata el clustering como un problema de optimización combinatoria y demuestra ser eficiente optimizando la función objetivo propuesta.

1. Introducción

En Ingeniería de Software, en las líneas de investigación de reutilización e ingeniería reversa, los componentes de software, como: programas, rutinas, procedimientos, etc. deben ser estructurados y reutilizados por diferentes sistemas. En esa perspectiva, los mecanismos de agrupamiento, tipo clustering, han jugado un papel importante en el desarrollo de técnicas para el particionamiento, la recuperación y reestructuración de software [8].

La recuperación es uno de principales problemas que se presenta en la Ingeniería de Software, el objetivo es obtener un modelo del sistema que permita lograr un mejor entendimiento de como este se encuentra estructurado. La búsqueda de estos grupos se realiza de tal manera que se satisfaga los criterios de cohesión y acoplamiento. Es decir, los grupos a encontrar deben tener un alto grado de cohesión, y bajo acoplamiento con respecto a otros grupos.

Estos objetivos concuerdan con los del clustering, donde se busca obtener grupos que sean lo mas similares entre si, y a la vez diferente de otros clusters. Witggerts [18] les denomina entidades a los objetos a agrupar y estos pueden ser programas, funciones o procedimientos.

Existen dos formas de representación del software para recuperar su estructura. La primera es a través del uso de grafos, donde los nodos representan las entidades, y los vertices las relaciones que existen entre ellos. El objetivo es encontrar grupos que contengan nodos altamente relacionados (alta cohesión) y que tengan menos relaciones posibles entre ellos (bajo acoplamiento), estos grupos serán los módulos o subsistemas del software en análisis. Debido a que este

es un problema NP-Difícil, se han propuesto diversos métodos heurísticos y metaheurísticos para encontrar un solución eficiente del problema [12, 4].

La segunda forma, y aquí es donde encaja el presente trabajo, es a través de vectores que contenga las características que presenta la entidad de software. Estas pueden ser referencias a variables, llamadas a otros módulos o tipos de accesos. La representación en este caso es mediante vectores binarios, en donde 1 indica la presencia de la característica y 0 su ausencia. Mediante técnicas de clustering se buscan encontrar grupos que contengan entidades que sean lo mas similares entre si (alta cohesión) y lo mas diferentes de otros grupos (bajo acoplamiento). Cuando se trata de sistemas legados, que fueron elaborados sin previa documentación de diseño, obtener esta abstracción de alto nivel es realmente importante porque ayudará a los ingenieros de software a tener un mejor entendimiento de la arquitectura del sistema cuando se necesiten realizar modificaciones al sistema. Este problema es también NP-Difícil, y en la literatura podemos encontrar diversos métodos de clustering, clasificados como jerárquicos, para encontrar una solución al problema [18, 13, 15, 10]. Estos métodos tienen la desventaja de que implican un alto costo computacional en la búsqueda de soluciones [7]. Es aquí, donde se hace necesario la propuesta de un método de clustering de optimización basado en centros. Donde, los centros son aquellos elementos que mejor representan al cluster, y por ello también le denominaremos elemento representativo.

El problema del clustering consiste en encontrar grupos de objetos que sean similares entre sí y a la vez diferentes a los objetos pertenecientes a otros grupos, de tal manera que se satisfaga un criterio establecido. Los grupos con características similares son conocidos como *clusters*. Los objetos son representados por vectores en el espacio R^D, y con el uso de una medida de la similaridad, como la distancia, se definen los clusters. No existe conocimiento previo acerca de cómo se debe definir un cluster; por tal motivo, el proceso de clustering es conocido como clasificación no supervisada.

El clustering tiene múltiples aplicaciones dentro de las ciencias de la computación, como compresión de imágenes [16] y voz digitalizadas [9]; en la recuperación de información [1]; en minería de datos [5]; en la segmentación de imágenes médicas [14], en la clasificación de componentes de software [8], y otros.

En este trabajo proponemos un mecanismo de agrupamiento de entidades de software, estableciendo una cuantificación de los atributos de los componentes de forma que puedan ser usado por técnicas de clustering basado en KMeans en el marco de la metaheurística GRASP, técnica puramente numérica. Ese mecanismo va a permitir seleccionar elementos representativos de cada grupo de un conjunto grande de componentes de software.

Resto del trabajo está organizado de la siguiente manera: en la Sección 2 se hace una presentación de Clustering y revisión de los trabajos recientes, así como la presentación de la metaheurística GRASP. En la Sección 3 se presenta la adaptación del método Grasp-KM para el clustering de software. En la Sección 4 se muestran los experimentos computacionales realizados. Finalmente, en la Sección 5 se exponen las conclusiones y trabajos futuros.

2. Antecedentes

En esta sección se hace una revisión de los principales conceptos para la aplicación de las técnicas de clustering en la ingeniería de software. Primero se define el problema del clustering y se revisan las medidas de similaridad empleadas para el clustering de software. Luego, se hará una revisión los trabajos existentes en la literatura. Finalmente, se hace una revisión de la metaheurística GRASP.

2.1. Problema de Clustering

Dado un conjunto de n objetos denotado por $X = \{x_1, x_2, \ldots, x_n\}$, en que $x_i \in R^D$. Sea K un número entero positivo, el problema del clustering consiste en encontrar una partición $P = \{C_1, C_2, \ldots, C_K\}$ de X, siendo C_j un cluster definido por objetos similares, satisfaciendo una función objetivo $f : R^D \rightarrow R$ que representa la distancia mínima entre los objetos del cluster, y las condiciones:

$$C_i \bigcap C_j = \emptyset \text{ para } i \neq j, \quad \text{y} \quad \bigcup C_i = X.$$

La definición de la función de similaridad depende de la naturaleza de los objetos a agrupar.

2.2. Medidas de similaridad

Diversas medidas han sido propuestas para medir la similaridad entre objetos. Wiggerts[18], clasifica las medidas de la similaridad en cuatro tipos: medidas de la distancia, coeficientes de asociación, coeficientes de correlación y medidas probabilísticas. Son destacados los dos primeros tipos de medidas. Los coeficientes de asociación se usan principalmente para medir la similaridad entre dos vectores binarios, obteniendo valores en el intervalo $[0, 1]$; cuanto más cercano a 1, indica que los vectores son similares entre sí. Por otro lado, las medidas de la distancia obtienen como resultado un

valor real positivo entre dos entidades, donde un valor cercano a 0 indicará que los objetos son similares.

Medidas de la distancia

La noción de similaridad entre dos entidades representadas por dos vectores x y $y \in R^D$, es caracterizada por una función distancia como

$$d : (x, y) \rightarrow R.$$

Se dice que dos objetos $x = (x_1, ..., x_D)$ y $y = (y_1, ..., y_D)$ son similares cuando la distancia entre los vectores es menor que una tolerancia pequeña. En [2] se describen las propiedades que debe tener la distancia.

La elección de una función distancia entre los vectores depende del grado de dificultad de las entidades y su interpretación. La más usada es la distancia Euclidiana, definida por:

$$d_2(x, y) = \sqrt{\sum_{h=1}^{D} (x_h - y_h)^2}.$$

La distancia Euclidiana es un caso especial de la medida de Minkowski cuando $p = 2$, dada por

$$d_p(x, y) = \sqrt[p]{\sum_{h=1}^{D} (x_h - y_h)^p}.$$

Coeficientes de asociación

Los coeficientes de asociación, o de comparación, para dos entidades $E1$ y $E2$ representadas por vectores binarios x y y, respectivamente, están dadas por el número de atributos coincidentes de una entidad en relación a otra. Lung et al. [8] clasifica a este tipo de coeficientes como cualitativos, debido a que calculan la similaridad basado en la ausencia o presencia de atributos. Según Wiggerts [18], son cuatro casos de asociación entre las entidades respecto al número de sus atributos: presentes en ambas entidades (a); presentes en $E1$ pero no en $E2$ (b); presentes en $E2$ pero no en $E1$ (c); y no presentes en ambos (d). Si se denota por 1 binario como presencia de un atributo en una entidad, y por 0 la ausencia, es apropiado relacionar la ocurrencia de esos atributos por una tabla definida como:

		$E2$	
		1	0
$E1$	1	a	b
	0	c	d

Por ejemplo, sean dos entidades $E1$ y $E2$, descritas através de dos vectores binarios $x = (0, 1, 0, 1, 1, 1)$ y $y = (0, 1, 1, 0, 1, 0)$, respectivamente. Entonces, $a = 2$ porque los atributos presentes $(1 - 1)$ están en la segunda y quinta posición de ambos vectores. El valor de $b = 2$ porque los atributos cuarto y sexto están en x pero no en y, caso $(1 - 0)$. Así, se observan que $c = 1$, para $(0 - 1)$ y $d = 1$ para $(0 - 0)$.

Existen diversos métodos para calcular los coeficientes de asociación; ellos se basan, principalmente, en la relevancia de las coincidencias entre ambos vectores y la ponderación que le asignan. Los principales métodos para el cálculo de coeficientes entre dos vetores x y y, usados en [15, 18, 8], son:

- Coeficiente de **Jaccard**: $S_j(x, y) = a/(a + b + c)$
- Coeficiente **Simple**: $S_s(x, y) = (a+d)/(a+b+c+d)$
- Coeficiente de **Sorensen**: $S_r(x, y) = 2a/(2a+b+c)$

Se observa que el coeficiente de Jaccard y Sorensen considera relevantes las relación $1 - 1$, pero no las relación $0 - 0$ ya que estas indican la ausencia de atributos. El coeficiente Simple, considera relevantes tanto las relaciones $1 - 1$, como las $0 - 0$.

En [13], se propone una medida de similaridad en función de producto y norma de vectores binarios x y y, la cual también es usada para calcular la similaridad entre documentos por métodos de recuperación de información. La medida esta dada por la siguiente expresión:

$$S_1(x, y) = \frac{x \times y}{\|x\|\|y\|}.$$

En el mismo trabajo, se extiende esta función de medida a vectores no binarios, donde los valores expresan la frecuencia de ocurrencia de cierta característica. Por ejemplo, si $x = (x_1, \ldots, x_n)$ y $y = (y_1, \ldots, y_n)$ representan a dos entidades de software (i.e. programas), entonces los valores x_i y y_i pueden expresar la cantidad de veces que datos tipo T_i son declarados en dichos programas. La función de medida extendida envuelve producto interno de vectores,

$$S_2(x, y) = \frac{x \cdot y}{\|x\|\|y\|}.$$

2.3. Software Clustering

En [15] se presenta un método para agrupar entidades de software representadas por vectores binarios. Una de las características relevantes del método es, que a partir de los vectores que conforman un cluster se obtiene un vector representativo. Este vector se obtiene aplicando el operador OR a los vectores binarios que conforman el cluster. Por ejemplo,

si los vectores $x_1 = (0,1,1)$ y $x_2 = (0,0,1)$ conforman el cluster C entonces, el vector representativo del cluster será $x_C = (0,1,1)$. Los clusters se van construyendo a través del método jerárquico *Weighted Average Linkage*, y la medida de similaridad usada es el coeficiente de asociación Jaccard. El método propuesto tiene el inconveniente de que al aplicar el operador OR para obtener el vector representativo del cluster, se ignora la cantidad de entidades que presentan la misma característica. Por ejemplo, si se tienen los clusters $A = \{(0,1,0),(0,1,0),(1,1,0)\}$ y $B = \{(0,0,1),(0,1,1)\}$, entonces los vectores representativos de A y B son $x_A = (1,1,0)$ y $x_B = (0,1,1)$, respectivamente.

Al calcular el coeficiente de asociación de un vector $x_6 = (0,1,0)$ a los clusters A o B se obtendría el mismo valor, y se podría asignar a cualquiera de ellos. Sin embargo se observa que el cluster A, tiene mas vectores con el valor $x_{i2} = 1$, y por tanto lo lógico sería que se asocie al cluster A.

En [10] se presenta una mejora al método propuesto en [15]. El cálculo del vector representativo se basa en la frecuencia con que se presentan las características en las entidades que componen el cluster. Es decir para el caso anterior: el vector representativo del cluster A, será $x_A = (1/3, 3/3, 0/3)$ y para el cluster B, será $x_B = (0/2, 1/2, 2/2)$. Esto quiere decir que el elemento representativo de un cluster $C = \{(x_{11},\ldots,x_{1d}),\ldots,(x_{n1},\ldots,x_{nd})\}$ estará dado por

$$x_C = \left(\frac{\sum_{i=1}^{n} x_{i1}}{n},\ldots,\frac{\sum_{i=1}^{n} x_{id}}{n}\right). \qquad (1)$$

El vector representativo deja de ser binario y por tanto ya no se puede aplicar los coeficientes de asociación revisados en la punto antrerior. El autor extiende la medida de similaridad denominada *Ellenberg* para tomar en cuenta las frecuencias de las características. A la medida de la similaridad propuesta le denomina *unbiased Ellenberg*, y esta dada por la siguiente expresión:

$$S_e(x,y) = \frac{(0{,}5)M_a}{(0{,}5)M_a + b + c} \qquad (2)$$

Donde, M_a representa la suma de las características que están presentes en ambas entidades, y b y c representan la cantidad de características que están presentes en una entidad y no en la otra. Con esta medida de la similaridad, en [10] se utilizan métodos jerárquicos de clustering mejorando los resultados obtenidos en [15].

2.4. Metaheurística GRASP

Un procedimiento de búsqueda voraz aleatoria y adaptativa (GRASP) es una metaheurística propuesta por Feo y Resende [6] para encontrar soluciones aproximadas de problemas de optimización combinatoria, mediante un proceso iterativo. En cada iteración se realizan los procesos de *construcción* y *búsqueda local*. En la construcción se genera un conjunto solución S de una instancia E de un problema combinatorio, y en la búsqueda local se determina una posible mejor solución a S; finalmente, se elige la mejor solución entre la solución de la iteración anterior y la actual. La mejor solución será evaluada por una función objetivo f. Todo el proceso es repetido un número máximo de veces (MAX_ITER). El Algoritmo 1 muestra el marco general de la meteheurística GRASP.

Algoritmo 1: Grasp

entrada: E, MAX_ITER, α

```
1  inicio
2      Inicializar solución S := ∅ y f* := ∞ ;
3      para i = 1 hasta MAX_ITER hacer
4          S* := Construccion_Grasp(E, α) ;
5          S* := Busqueda_Local_Grasp(S*) ;
6          si f(S*) < f* entonces
7              Actualizar S := S* y f* := f(S*) ;
8          fin
9      fin
10     retornar S
11 fin
```

El hecho de que la búsqueda local toma como entrada la solución obtenida en la construcción proporciona un conocimiento frente a los algoritmos de búsqueda local tradicionales.

Construcción GRASP

En esta fase se adapta un algoritmo goloso que selecciona el mejor elemento de un conjunto de candidatos C ha ser incorporados en la solución. El criterio de selección goloso depende del problema, puede ser de maximización o minimización. El constructor de soluciones evita el determinismo de los algoritmos golosos, utilizando un parámetro de relajación $\alpha \in [0,1]$ para formar una lista restringida de candidatos (*Restricted Candidate List* - RCL) alrededor del mejor elemento a seleccionar. El elemento ha ser incorporado en la solución es elegido aleatoriamente del RCL. Esta forma estocástica de selección, permite construir soluciones con tendencia a los mejores elementos y evita caer en

óptimos locales. El parámetro de relajación α indica la amplitud del RCL alrededor del mejor candidato. El mejor valor de α para el problema en estudio se obtiene a través de múltiples experimentos computacionales de calibración.

Búsqueda Local GRASP

La búsqueda local se realiza de manera iterativa, explorando en la vecindad de un conjunto de solución S, generada en la construcción. El desempeño de la operación de búsqueda local dependerá del método elegido. Si N es una vecindad de soluciones, se dice que $S' \in N(S)$ es un óptimo local si $f(S') < f(S)$. No existe un esquema de búsqueda local específico a utilizarse, solo es necesario que mejore la solución encontrada en la construcción.

Algoritmo 2: GraspKM

entrada: X, K, α, MAX_ITER

```
1  inicio
2      f* := ∞, C := {} ;
3      para i = 1 hasta MAX_ITER hacer
4          C' := InicializacionKM(X, K) ;
5          C' := ConstruccionKM(X, K, C', α) ;
6          C' := MejoriaKM(X, K, C') ;
7          si f(C') < f* entonces
8              C:= C' ;
9              f*:= f(C') ;
10         fin
11     fin
12     retornar C
13 fin
```

3. Software Clustering usando GRASP

En [17] se propone el método GraspKM para encontrar una solución viable al problema del hard clustering, es decir, cuando los grupos encontrados son particiones del conjunto de objetos en análisis. El método GraspKM es una adaptación del algoritmo KMeans [11] dentro del marco de la metaheurística GRASP.

El algoritmo KMeans construye los clusters iterativamente, partiendo de K posibles centros seleccionados aleatoriamente de un conjunto X de N elementos. Los clusters se definen asignando cada elemento de X de forma que la distancia respecto al posible centro sea mínima respecto a otros centros. En cada iteración siguiente, se recalculan los centros de los clusters como la media aritmética de los elementos que lo componen; y si las variaciones de los centros aun persisten, entonces se vuelve a iterar hasta que los centros no varíen.

El GraspKM, mostrado en el Algoritmo 2, inicia de manera similar al KMeans. Se eligen aleatoriamente K objetos como centros y se forman los clusters iniciales asignando cada uno de los objetos al cluster cuyo centro se encuentre más cercano. Luego, se asignan iterativamente cada uno de los objetos a un cluster elegido aleatoriamente de una RCL. Este proceso se repite hasta que no haya mas reasignaciones. Finalmente, se obtiene una mejor solución a través de un algoritmo de búsqueda local.

El método utiliza como medida de similaridad la distancia euclidiana y demuestra ser superior al algoritmo K-Means y comparable con otras metaheurísticas. Aunque este método es eficiente encontrando soluciones para objetos representados por vectores en R^d, este no puede ser aplicado directamente al clustering de software y debe ser adaptado para cubrir los siguientes puntos:

- Las medidas de la similaridad en el clustering de software no están basadas principalmente en la distancia, sino en los coeficientes de asociación debido a que se trabaja con vectores binarios.
- Los coeficientes de asociación permiten el calculo de la similaridad entre dos vectores y entre grupo de vectores, pero la obtención de un vector centro que represente al grupo, no es de manera natural, como si lo permite la distancia euclidiana.

En esta perspectiva, se debe adaptar el método propuesto en [17] para agrupar vectores binarios que representan a las entidades de software. Como se describió en la sección anterior, en [10] se propone una medida de la similaridad llamada *unbiased Ellenberg* (2), que toma en cuenta la frecuencia con que se presentan las características en los componentes de software. Esta medida obtiene un valor entre $[0, 1]$, donde una valor de 1 o cercano indica que son similares. Para poder formular la función objetivo respecto a la minimización, se requiere tener una medida que cuando sea similar se acerque a cero. Por tanto, la medida de la similaridad que usaremos es:

$$S_m(x, y) = 1 - \frac{(0{,}5)M_a}{(0{,}5)M_a + b + c}. \qquad (3)$$

Esta medida permitirá calcular la similaridad entre el elemento representativo del cluster y un vector binario. En el referido trabajo [10], se propone uso de un elemento representativo para el clustering de software, aunque este no es un método de clustering basado en

centros como el algoritmo KMeans, es posible usar este elemento como centro, incluso estos coinciden con los centros usados en el algoritmo KMeans. Por tanto, los elementos representativos que usaremos para el algoritmo GraspKM estarán dados por la expresión (1).

Como lo que se quiere es obtener clusters con entidades de software que sean lo mas parecidas entre si. Entonces podemos definir empíricamente la similaridad de un cluster C como:

$$S(C) = \sum_{x \in C} S_m(x, \bar{x}_c). \quad (4)$$

En base a esta expresión podemos formular la función objetivo f como:

$$f = \sum_{j=1}^{K} S_m(C_j), \quad (5)$$

donde K es el número de clusters que deseamos definir y el objetivo del método a desarrollar debe ser minimizar esta función.

En la fase InicializacionKM, se eligen aleatoriamente K vectores de X como centros, y conforma los clusters iniciales asociando cada uno de los elementos de X al cluster mas cercano. Luego, se recalculan nuevamente los elementos representativos.

Como los elementos representativos han variado, los objetos deben ser asignados nuevamente a los clusters más cercanos. Esto se hace a través del proceso iterativo denominado ConstruccionKM, tal como es mostrado en el Algortimo 3, donde los posibles clusters que contendrían al objeto x en análisis, son agrupados en un conjunto RCL cuyas medidas de similaridad entre el cluster y el objeto x están en un intervalo definido por $\overline{\beta}$ y $\underline{\beta}$ y regulada linealmente por el parámetro de relajación α. Del conjunto RCL será elegido aleatoriamente un cluster al cual será reasignado el objeto x, y retirándolo del cluster donde se encontraba previamente. Después de la reasignación de todos los objetos de X los elementos representativos han variado; por tanto, nuevamente deben ser recalculados. El proceso termina cuando los elementos representativos no varían.

Las soluciones obtenidas en la fase de construcción son refinadas en la fase denominada MejoriaKM, que es un proceso iterativo de dos fases: ReagrupacionKM y ConstruccionKM, que se repiten hasta que la solución no pueda ser mejorada. La idea de la reagrupación es eliminar y generar nuevos cluster heurísticamente. Se elimina y se genera un nuevo clusters a la vez; el cluster a eliminar es aquel que presenta la menor cantidad de objetos, y se divide aquel cluster que tiene la mayor dispersión, esto debido a que en ambos casos el proceso puede haber caído en un óptimo local. Si C_l es el cluster con menor número de elementos, entonces l esta dado por:

$$l = ArgMin\{|C_j|\}_{j=1,...,K}. \quad (6)$$

El calculo del cluster C_h con mayor dispersión esta determinado por:

$$h = ArgMax \left\{ \frac{Sim(C_j)}{|C_j|} \right\}_{j=1,...,K, j \neq l}. \quad (7)$$

Donde, $Sim(C_j)$ es la similaridad del cluster C_j y es calculada usando la expresión (4). Luego de que es eliminado el cluster y generado uno nuevo, cada uno de los objetos de X son asignados a los nuevos centros. Finalmente, los clusters obtenidos son refinados con el proceso ConstruccionKM.

Algoritmo 3: ConstruccionKM

entrada: X, K, C, α

inicio
  repetir
    para cada $x \in X$ *tal que* $x \in C_{j=1,...,K}$ **hacer**
      $\overline{\beta} := Max\{S_m(x, \bar{x}_l) :$
      $S_m(x, \bar{x}_l) \leq S_m(x, \bar{x}_j)\}_{l=1,...,K}$;
      $\underline{\beta} := Min\{S_m(x, \bar{x}_l)\}_{l=1,...,K}$;
      $RCL := \{C_t :$
      $S_m(x, \bar{x}_t) \leq \underline{\beta} + \alpha(\overline{\beta} - \underline{\beta})\}_{t=1,...,K}$;
      $C_t := Random(RCL)$;
      si $t \neq j$ **entonces**
        $C_t := C_t \cup \{x\}$;
        $C_j := C_j - \{x\}$;
      fin
      Recalcular elementos representativos;
    fin
  hasta *No se realicen reasignaciones* ;
  retornar $C = \{C_j\}_{j=1,...,K}$
fin

4. Experimentos Computacionales

En [3] se presenta un método para la identificación de módulos de un sistema basado en reglas de asociación. En este trabajo se utiliza, para la comprobación del método, un conjunto de datos que consiste en 28 programas escritos en COBOL, definidos como el conjunto $P = \{p_1, p_2, ..., p_{28}\}$, los cuales usan 36 archivos de datos $F = \{f_1, f_2, ..., f_{36}\}$. En el referido trabajo se utiliza la metodología ISA (*Identification of Subsystems based on Associations*) para identificar los

subsistemas basados en asociaciones. Esta metodología realiza como primer paso, una selección de los datos que considera mas relevantes para el proceso. El resultado de esta selección se le conoce como *AlphaSet*, y consiste en seleccionar aquellos programas que usen más de un valor γ de archivos, y seleccionar los archivos que usen más de un valor β de programas. Ambos parámetros deben ser enteros positivos y para el caso se usan los valores: $\gamma = 1$ y $\beta = 1$. Luego de este proceso previo, se obtiene un subconjunto $\tilde{P} \subset P$ de 22 programas y un subconjunto $\tilde{F} \subset F$ de 24 archivos de datos. Esas informaciones son procesadas por el método GraspKM, adaptado al clustering de software, para identificar los módulos del sistema, agrupando aquellos programas que acceden a archivos de datos similares. En el Cuadro (1) se muestran los datos a usar.

Nro.	Programas ($\tilde{P}$)	Archivo de datos usados ($\tilde{F}$)
1	p_1	f_3, f_5
2	p_2	f_3, f_5
3	p_5	f_3, f_5, f_{26}
4	p_6	f_3, f_5, f_{26}
5	p_8	f_3, f_5, f_{26}
6	p_9	f_3, f_5
7	p_{10}	f_3, f_5
8	p_{13}	f_3, f_5, f_{26}
9	p_{14}	f_3, f_5, f_{26}
10	p_{15}	f_3, f_5, f_{26}
11	p_{16}	$f_9, f_{10}, f_{12}, f_{18}, f_{19}, f_{22}, f_{23}, f_{24}, f_{26}, f_{27}, f_{29}, f_{30}, f_{32}, f_{33}, f_{34}, f_{35}, f_{36}$
12	p_{17}	f_{26}, f_{30}
13	p_{18}	$f_9, f_{10}, f_{12}, f_{17}, f_{18}, f_{19}, f_{22}, f_{23}, f_{24}, f_{25}, f_{26}, f_{27}, f_{29}, f_{32}, f_{33}, f_{34}, f_{35}, f_{36}$
14	p_{19}	$f_{10}, f_{12}, f_{17}, f_{19}, f_{22}, f_{23}, f_{24}, f_{25}, f_{26}, f_{27}, f_{29}, f_{32}, f_{33}, f_{34}, f_{35}, f_{36}$
15	p_{20}	$f_{14}, f_{23}, f_{29}, f_{32}$
16	p_{21}	f_5, f_{14}
17	p_{23}	$f_3, f_5, f_{23}, f_{26}, f_{27}, f_{28}$
18	p_{24}	f_3, f_5, f_{26}
19	p_{25}	f_{20}, f_{23}, f_{26}
20	p_{26}	f_3, f_{23}, f_{26}
21	p_{27}	f_{20}, f_{23}, f_{26}
22	p_{28}	$f_3, f_5, f_{23}, f_{26}, f_{28}$

Cuadro 1. Conjunto de programas a agrupar.

Cada uno de los 22 programas será representado con un vector binario de dimensión 24, donde el valor de 1 indicará el uso del archivo de datos en el orden respectivo.

Para determinar el valor apropiado para el parámetro α, se realizaron experimentos con un valor de $MAX\_ITER = 1,000$ para distintos valor de α. Los mejores resultados obtenidos para la función objetivo, expresión (5), se presentan en el Cuadro (2). El cuadro muestra que con un valor de $\alpha = 1$ se obtienen los mejores resultados, este valor es el mismo encontrado en las experiencias numéricas realizados en [17]. Es necesario resaltar que para con el valor de $\alpha = 1$, el método GraspKM no se comporta como un aleatorio puro, debido a las restricciones impuestas en la fase ConstruccionKM.

α	0	0.25	0.50	0.75	1.00
K = 3	10.237	10.237	10.237	10.237	7.028
K = 4	5.684	5.684	5.684	5.684	5.449

Cuadro 2. Calibración de parámetro α.

Con estos parámetros, compararemos los resultados obtenidos por el método GraspKM y el algoritmo KMeans adaptado también al clustering de entidades de software. La comparación se realiza en cuanto a la eficiencia para obtener la función objetivo f. Para el algoritmo KMeans se consideran $1,000$ ejecuciones y se considera el mejor valor obtenido. Los resultados se muestran en el siguiente Cuadro (3) y como se puede apreciar, el método GraspKM encuentra mejores valores de f para el conjunto de datos usado.

	KMeans	GraspKM
K = 3	10.237	7.028
K = 4	7.571	5.449

Cuadro 3. Valor de f obtenido por KMeans y GraspKM.

Los clusters encontrados por el método GraspKM cuando $K = 3$ y $K = 4$ se muestran en las cuadros 4 y 5. En ambos casos se muestra la configuración de clusters del mejor resultado obtenido para f con los parámetros de $MAX\_ITER = 1000$ y $\alpha = 1$.

Clusters	Programas
C_1	$p_1, p_2, p_5, p_6, p_8, p_9, p_{10}, p_{13}, p_{14}, p_{15}, p_{17}, p_{21}, p_{24}, p_{26}$
C_2	$p_{16}, p_{18}, p_{19}, p_{20}$
C_3	$p_{23}, p_{25}, p_{27}, p_{28}$

Cuadro 4. Clusters para $K = 3$.

Clusters	Programas
C_1	$p_1, p_2, p_5, p_6, p_8, p_9, p_{10}, p_{13}, p_{14}, p_{15}, p_{21}, p_{24}, p_{26}$
C_2	$p_{16}, p_{18}, p_{19}, p_{20}$
C_3	p_{17}, p_{25}, p_{27}
C_4	p_{23}, p_{28}

Cuadro 5. Clusters para $K = 4$.

Como se puede apreciar en los resultados, los clusters se encuentran definidos por programas que acceden a similares archivos de datos, lo cual nos da una idea de la estructura del sistema respecto a los datos que maneja.

5. Conclusiones y trabajos futuros

En el presente trabajo se adapta la metaheurística GRASP para el clustering de software. El método de-

nominado GraspKM , que aborda el problema del clustering como de optimización combinatoria y encuentra una solución eficiente basado en los centros, es adaptado en lo que respecta al uso una medida de la similaridad propia de vectores binarios y al uso de un vector representativo de los clusters. En las pruebas numéricas, el método demuestra ser superior que el algoritmo KMeans adaptado para el clustering de software. Esta método permite encontrar grupos de entidades de software que presenten características similares (cohesión) y a la vez diferentes de otros grupos (bajo acoplamiento), optimizando la función objetivo f propuesta.

Al igual que los métodos de clustering revisados en la literatura, el valor de K es un parámetro que debe ser ingresado. En este sentido, se puede extender el presente trabajo de manera que el valor de K puede ser determinado de manera automática.

El método propuesto puede ser extendido para su uso en el área de recuperación de información, donde se necesita encontrar grupos de documentos similares. Estos documentos generalmente se encuentran representados por vectores binarios donde 1 indica la presencia de cierta palabra en el documento; o vectores que contienen la frecuencia de ocurrencia de cierta palabra en el documento. En ambos casos, el método propuesto puede ser adaptado para su uso.

Referencias

[1] S. Bathia and J. Deogun. Conceptual clustering in information retrieval. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 28(3):427–436, 1998.

[2] E. Chavez, G.Ñavarro, R. Baeza-Yates, and J. Marroquín. Searching in metric spaces. *ACM Compunting Surveys*, 33(3):273–321, 2001.

[3] C. M. de Oca and D. L. Carver. Identification of data cohesive subsystems using data mining techniques. In *ICSM '98: Proceedings of the International Conference on Software Maintenance*, page 16, Washington, DC, USA, 1998. IEEE Computer Society.

[4] D. Doval, S. Mancoridis, and B. Mitchell. Automatic clustering of software systems using a genetic algorithm. In *IEEE Proceedings of the 1999 International Conference on Software Tools and Engineering Practice (STEP'99)*, Pittsburgh, PA, August 1999.

[5] U. Fayyad, G. Piatetsky-Shapiro, and P. S. From data mining to knowledge discovery in databases. *American Association for Artificial Inteligence*, pages 37–54, 1996.

[6] T. Feo and M. Resende. Greedy randomized adaptative search procedure. *Journal of Global Optimization*, 6:109–133, 1995.

[7] A. Jain, Murty, and M. F. P. Data clustering: a review. *ACM Computer Surveys*, 31(3):264–323, 1999.

[8] C.-H. Lung, M. Zaman, and A.Ñandi. Applications of clustering techniques to software partitioning, recovery and restructuring. *J. Syst. Softw.*, 73(2):227–244, 2004.

[9] J. Makhoul, S. Roucos, and H. Gish. Vector quantization in speech coding. *Pattern Recognition*, 73:1551–1558, 1985.

[10] O. Maqbool and H. A. Babri. The weighted combined algorithm: A linkage algorithm for software clustering. volume 00, page 15, Los Alamitos, CA, USA, 2004. IEEE Computer Society.

[11] J. McQueen. Some methods for classification and analysis of multivariate observations. *In Preccedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

[12] B. Mitchell and S. Mancoridis. Using heuristic search techniques to extract design abstractions from source code. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'03)*, Chicago, Illinois, July 2002.

[13] S. Patel, W. Chu, and R. Baxter. A measure for composite module cohesion. In *ICSE '92: Proceedings of the 14th international conference on Software engineering*, pages 38–48, New York, NY, USA, 1992. ACM Press.

[14] D. Pham and J. Prince. An adaptive fuzzy c-means algorithm for image segmentation in the presence of intensity inhomogeneities. *Pattern Recognition Letters*, 20(1):57–68, 1999.

[15] M. Saeed, O. Maqbool, H. Babri, S. Hassan, and S. Sarwar. Software clustering techniques and the use of combined algorithm. volume 00, page 301, Los Alamitos, CA, USA, 2003. IEEE Computer Society.

[16] P. Scheunders. A genetic lloyd-max image quantization algorithm. *Pattern Recognition Letters*, 17(5):547–556, 1996.

[17] E. Vicente, L. Rivera, and D. Mauricio. Grasp en la resolución del problema del clustering. In *CLEI 2005: XXXII Conferencia Latinoamericana de Informática*, Santiago de Cali, Colombia, 2005. CLEI.

[18] T. A. Wiggerts. Using clustering algorithms in legacy systems remodularization. In *WCRE '97: Proceedings of the Fourth Working Conference on Reverse Engineering (WCRE '97)*, page 33, Washington, DC, USA, 1997. IEEE Computer Society.

Testing exploratorio en la práctica

Beatriz Pérez, Amparo Pittier, Mariana Travieso, Mónica Wodzislawski
Centro de Ensayos de Software,
Universidad de la República,
Instituto de Computación,
Montevideo, Uruguay
{ bperez, apittier, marianat, mwodzis }@fing.edu.uy

Resumen

El testing exploratorio se define como el aprendizaje, el diseño de casos de prueba y la ejecución de las pruebas en forma simultánea [1]. Dentro de los servicios que brinda el Centro de Ensayos de Software (CES) está la prueba funcional independiente de productos de software desarrollados por terceros. En este artículo se describe la experiencia en el CES de realizar la prueba independiente de un producto de software donde se contaba con un plazo muy corto de tiempo. Es por esto que se decidió utilizar la estrategia de testing exploratorio.

Se describe la estrategia de planificación y el abordaje de testing exploratorio utilizado, se muestran los resultados obtenidos y se concluye sobre las ventajas y desventajas de este enfoque.

1. Introducción

El término "testing exploratorio" fué introducido por Cem Kaner, se refiere a ejecutar las pruebas a medida que se piensa en ellas, sin gastar demasiado tiempo en preparar o explicar las pruebas, confiando en los instintos. El testing exploratorio se define como el aprendizaje, el diseño de casos de prueba y la ejecución de las pruebas en forma simultánea. En otras palabras, es una técnica de prueba en la cual quien prueba controla activamente el diseño mientras son realizadas, y utiliza la información obtenida en la exploración para diseñar nuevas y mejores pruebas [1]. En el testing exploratorio siempre se debe tomar nota de lo que se hizo y lo que sucedió [2]. Los resultados del testing exploratorio no son necesariamente diferentes de aquellos obtenidos de la prueba con diseño previo y ambos enfoques para las pruebas son compatibles [3].

El testing exploratorio puede ser aplicado en cualquier situación donde no sea obvio cuál es la próxima prueba que se debe realizar. También es adecuado cuando se requiere obtener realimentación rápida de cierto producto o funcionalidad, se necesita aprender el producto rápidamente, se quiere investigar y aislar un defecto en particular, se quiere investigar el estado de un riesgo particular, o se quiere evaluar la necesidad de diseñar pruebas para esa área.

Se presenta en este artículo la estrategia de testing exploratorio utilizada para probar un producto de software. La prueba fue realizada por el equipo del Centro de Ensayos de Software (CES), la empresa que desarrolló el producto le contrató al CES la tercerización de las pruebas del producto.

En la sección 2 se describen los conceptos y las distintas estrategias para el testing exploratorio.

En la sección 3 se describe el contexto donde se realizó la prueba, se presenta el Centro de Ensayos de Software y se describe el producto a probar.

En la sección 4 se presenta la estrategia utilizada para probar el producto, se describe la planificación del proyecto de prueba y la estrategia de testing exploratorio aplicada.

En la sección 5 se presentan los resultados obtenidos con la prueba exploratoria. Por último en la sección 6 se presentan las conclusiones.

2 Testing exploratorio

Una estrategia básica para realizar testing exploratorio es concebir un plan de ataque general, pero que permita desviarse de él por periodos cortos de tiempo e intereses diversos. Cem Kaner lo denomina el principio del "tour bus", las personas bajan del bus y conocen los alrededores con visiones variadas. La clave es no perderse el tour entero [1].

El éxito o el fracaso del testing exploratorio está íntimamente ligado con lo que sucede en la mente del tester. Algunas habilidades deseables en los testers son la capacidad de analizar el producto y evaluar los riesgos, utilizar herramientas y tener un pensamiento crítico acerca de lo que se sabe al momento de diseñar las pruebas. También es importante que los testers tengan la capacidad de distinguir lo observado de lo inferido; ya que las inferencias podrían inducirlos a no realizar pruebas que evidencien vulnerabilidades del producto. El uso de heurísticas desempeña un rol importante en la producción de ideas variadas. Una heurística es un mecanismo mental para generar pruebas variadas y acordes a las características del producto que se está probando, por lo que es deseable que los testers las tengan presentes al momento de realizar testing exploratorio [1].

En general, el testing meramente exploratorio requiere testers con mucha experiencia. Como ventaja se encuentra que es barato y rápido, como inconveniente, que, según algunos autores, produce escasa documentación y no facilita las medidas de cubrimiento[4].

El testing exploratorio presenta una estructura externa fácil de describir. Durante un período de tiempo un tester interactúa con un producto para cumplir una misión y reportar los resultados. Una misión describe qué se probará del producto, los tipos de incidentes que se buscan y los riesgos involucrados. La misión puede ser elegida por el tester o serle asignada por el líder de testing.

Los elementos externos básicos son: tiempo, tester, producto, misión y reportes. [1]. Esta aparente simplicidad permite desplegar una amplia gama de posibilidades para la aplicación del testing exploratorio.

2.1 Estrategias de aplicación

Desde el "estilo libre", aplicable en las primeras etapas de construcción de un producto, cuando los requerimientos, especificaciones y funcionalidades son aún ambiguos e inestables, se puede transitar hacia formas más estructuradas y documentadas del testing exploratorio.

En la selección de las distintas estrategias para su aplicación juegan un rol preponderante las necesidades de gerenciar y medir el proceso de testing exploratorio, de formalizarlo en mayor o menor grado. El resultado de aplicar el estilo libre consiste únicamente en el reporte de los incidentes detectados. Si se aplica el testing exploratorio basado en sesiones, los resultados incluyen notas escritas sobre los pasos seguidos y las observaciones realizadas, que facilitan el aprendizaje, el gerenciamiento y la acumulación de conocimiento [1]. En la literatura surgen, además del testing exploratorio basado en sesiones otras modalidades que se describen brevemente a continuación [5].

2.1 Testing exploratorio basado en sesiones

La técnica "Session Based Test Management" de James Bach [6], consiste en organizar el testing exploratorio en sesiones documentadas adecuadamente. Una sesión de testing exploratorio comprende generalmente un itinerario, que se establece a partir de la misión y eventualmente, algunas de las heurísticas a ser usadas. Su principal ventaja radica en que a pesar de su bajo costo relativo, permite elaborar reportes de avance, registrar el itinerario seguido, gestionar y medir el proceso. Es además, adaptable y flexible. Estas características son especialmente importantes cuando se está haciendo testing independiente para un cliente. Su desventaja es que depende fuertemente de las habilidades y preparación de los testers.

2.2 Testing funcional parcial

Se usa para testear funcionalidades individuales inmediatamente luego de implementadas, con el objetivo de decidir sobre su conformidad con los requerimientos y concepciones reales del diseño. Permite una rápida retroalimentación a los desarrolladores en etapas tempranas del ciclo de desarrollo.

2.3 Testing exploratorio realizado por usuarios

En muchas organizaciones, los usuarios exploran si las diferentes funcionalidades se adecuan a los escenarios reales de su trabajo. Estos usuarios tienen además del conocimiento del negocio, roles y responsabilidades variadas, que determinan naturalmente misiones específicas para el testing exploratorio que desenvuelven, no es preciso simularlas.

2.4 Testing de humo exploratorio

Se utiliza para tener una visión global y rápida sobre el nivel de calidad de una nueva versión de un producto, liberada para probar, en particular cuando las actualizaciones se producen periódica y frecuentemente. Se recorre la lista de funcionalidades básicas para detectar defectos o cambios en las funcionalidades. Por otra parte se recorre la lista de las correcciones para verificar que realmente se hayan realizado, así como las mejoras para verificar su comportamiento desde la perspectiva del usuario final.

2.5 Testing de regresión exploratorio

Cuando existen fuertes restricciones de tiempo, recursos humanos o financieros para realizar un testing de regresión exhaustivo, el testing se concentra en las correcciones y mejoras desarrolladas. Se basa fuertemente en la experiencia del tester para explorar la posible introducción de nuevos defectos o el surgimiento de efectos negativos colaterales.

3. Contexto de su aplicación

En esta sección se presenta el Centro de Ensayos de Software y el producto a probar.

3.1. El Centro de Ensayos de Software

El Centro de Ensayos de Software (CES) [7] es un emprendimiento conjunto de la Universidad de la República de Uruguay (UdelaR) y de la Cámara Uruguaya de Tecnologías de la Información (CUTI), entidad que agrupa a la mayoría de las empresas productoras de software del país.

Los servicios que ofrece el CES incluyen

- Servicios de prueba independiente: Planificar, diseñar, coordinar y ejecutar pruebas de productos de software de manera efectiva y controlada, definiendo claramente el contexto y los objetivos.
- Consultoría: Asesorar a las organizaciones en la mejora de los procesos de prueba, definición de estrategias y automatización de las pruebas. Colaborar en la creación y consolidación de sus áreas de prueba.
- Capacitación: Elaborar e impartir programas de capacitación en la disciplina de testing según las necesidades de cada organización.

El CES se compone de dos laboratorios: el Laboratorio de Testing Funcional enfocado en la evaluación de productos desde el punto de vista funcional y el Laboratorio de Ensayos de Plataformas, donde se realizan pruebas de desempeño y se asiste a la industria para resolver problemas de funcionamiento en arquitecturas de hardware y software complejas.

3.2. Producto a probar

El producto que se probó es una aplicación web. Algunas funcionalidades de la aplicación habían sido probadas anteriormente por el CES, por lo que eran conocidas por integrantes del equipo de pruebas. La empresa de desarrollo que contrató el servicio de testing funcional del CES, requería, que en un mes, se hiciera una prueba completa de todas las funcionalidades del producto en una nueva plataforma y manejador de base de datos. Estos cambios aumentaban la probabilidad de que muchas funcionalidades tuvieran errores.

Se definió junto con el cliente que al comenzar el proyecto de prueba la empresa de desarrollo liberaría una versión del producto, la cual se probaría durante un mes. A medida que se encontraran incidentes se reportarían al equipo de desarrollo. para que pudiera hacer las correcciones correspondientes en paralelo. Al mes, el equipo de desarrollo liberaría una nueva versión con los incidentes ya corregidos. En esa segunda versión sólo se realizarían pruebas de regresión.

La única documentación del producto disponible era el manual de usuario, aún no actualizado para la versión bajo prueba.

4. Estrategia utilizada

Se presenta en esta sección la estrategia de planificación para probar el producto y el abordaje de testing exploratorio utilizado.

4.1. Estrategia de planificación

El CES cuenta con un proceso definido para realizar pruebas funcionales independientes de productos. El proceso se llama ProTest [8] y es adaptado a cada proyecto de prueba. ProTest se basa en el análisis de riesgo del producto para definir la prioridad con que se van a realizar las pruebas. Se identifican las partes del sistema que en caso de fallar tienen las consecuencias más serias y aquellas que tienen mayor frecuencia de uso, ya que si una parte del sistema es usada frecuentemente y tiene un error, el uso frecuente hace que se tengan grandes posibilidades de que la falla aparezca. [9]

Excede el alcance de este artículo explicar en detalle el proceso seguido durante este proyecto de prueba, el cual puede ser consultado en [8], pero sí se explicará la estrategia de planificación utilizada.

En la Figura 1, se muestra la planificación del proyecto de prueba, donde se definió que la primera semana se dedicaría a la planificación y organización del proyecto de prueba, el primer ciclo de prueba llevaría 3 semanas y se tendría un segundo ciclo de 2 semanas donde se realizarían las pruebas de regresión. Debido a que el tiempo para las pruebas era muy corto y el producto tiene un gran número de funcionalidades, se trabajó con un equipo de pruebas de seis personas, dirigidas por un líder de pruebas. Al comenzar las pruebas se contaba sólo con dos personas del equipo de testing que conocían el producto. Durante la semana de planificación, los testers que no conocían el producto, lo exploraron asistidos por quienes tenían experiencia en el mismo. Se decidió que las personas con conocimiento en el producto diseñaran las sesiones de testing exploratorio, contestaran dudas respecto al funcionamiento de la aplicación y validaran los incidentes encontrados durante la ejecución de las pruebas, antes de incluirlos en el sistema de seguimiento de incidentes. Durante la semana de planificación se realizó junto con el cliente un inventario de las funcionalidades a probar, este inventario se realizó a partir de los menús y del manual de usuario de la aplicación. Se catalogaron 520 funcionalidades, se realizó el análisis de riesgo, dejando 55 funcionalidades fuera del alcance. Se planificó que en el ciclo de prueba 1 se probaran mediante testing exploratorio 465 funcionalidades. En el ciclo de prueba de regresión se verificaría que los incidentes encontrados fueron solucionados en la nueva versión del producto y pueden ser cerrados.

En la última semana del proyecto, se realizó la evaluación del mismo y el informe final del proyecto de prueba.

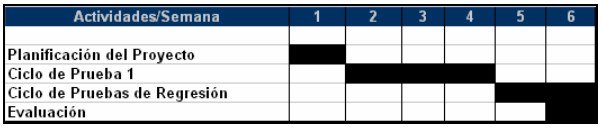


Figura 1 – Planificación del proyecto de prueba

4.2. Estrategia para Testing Exploratorio

Se optó por aplicar el testing exploratorio del producto basado en sesiones, dado lo exiguo de los plazos en relación a la cantidad de funcionalidades, y por el tipo de errores buscados.

Para definir las misiones, se estudiaron las funcionalidades de la aplicación y los ciclos funcionales. Se utilizaron dos estrategias. Por un lado, se definieron misiones en base a los principales ciclos funcionales de la aplicación y por otro, agrupando funcionalidades relacionadas. Las misiones estaban orientadas a probar que el cambio de plataforma y de manejador de base de datos no afectara el comportamiento esperado.

Las personas del equipo que conocían el producto diseñaron cinco misiones basadas en ciclos funcionales y diez misiones con funcionalidades relacionadas entre sí (por ejemplo, aquellas que pertenecían a un mismo menú o módulo), para que el resto de los integrantes del equipo llevaran a cabo sus sesiones. Luego, algunas de estas misiones fueron refinadas durante el proyecto.

Cada sesión se correspondía exactamente con una misión, y una misma misión podía ser objeto de varias sesiones.

Se definió como estrategia a seguir, que las misiones que cubrían las funcionalidades de mayor prioridad fueran asignados a más de una persona, lográndose así un cubrimiento más exhaustivo y rico.

A partir de las misiones asignadas a los integrantes del equipo de prueba, cada persona definió cada una de las sesiones que realizó. De esta forma, diseñó y ejecutó sus pruebas, registrando los resultados obtenidos en cada sesión y reportando en el sistema de seguimiento de incidentes los incidentes encontrados.

Se utilizó la herramienta Mantis [10] como sistema de seguimiento de incidentes. Dado que cuenta con una interfaz web, a medida que los incidentes eran reportados, el cliente los validaba y les asignaba la prioridad correspondiente. Esta dinámica resultó de vital importancia, ya que se logró que ambos equipos, el del CES y el de desarrollo del cliente, pudieran, en paralelo, probar y solucionar los incidentes detectados. De esta forma, la versión corregida para las pruebas de regresión, estuvo disponible casi inmediatamente, lo que permitió cumplir con los plazos establecidos.

Para cada incidente reportado se registraba la descripción, categoría, prioridad, ciclo de prueba en el cual era detectado y módulo. Mantis asigna un identificador único al incidente y registra el tester y la fecha de ingreso en forma automática.

Se definió una plantilla para registrar la información de la ejecución de las sesiones de testing exploratorio, con los siguientes datos:

- el ciclo de prueba correspondiente
- fecha y duración en minutos
- tester que realizó la ejecución
- misión de la sesión

- funcionalidades que fueron ejercitados al realizar la sesión
- razón por la que se ejecutó cada funcionalidad: por necesidad, por ser parte de la misión o por curiosidad
- datos de prueba
- observaciones: son aquellas cosas que llamaron la atención
- identificadores de los incidentes reportados en el sistema de seguimiento de incidentes.

Para poder controlar las pruebas que se estaban realizando y conocer el cubrimiento de funcionalidades que se iba obteniendo con el testing exploratorio, se mantuvo un registro de trazabilidad de las funcionalidades ya ejercitadas. Al finalizar cada jornada de trabajo, el líder de proyecto recopilaba la información de las funcionalidades ejercitadas durante las sesiones, incluyendo los incidentes encontrados, y actualizaba el documento de trazabilidad. En función de los resultados obtenidos en cada jornada, se definían las misiones para las siguientes sesiones. Esta realimentación constante fue guiando el foco de las pruebas a lo largo del proyecto.

5. Resultados obtenidos

En esta sección se exponen los resultados obtenidos en el proyecto. En la sección 5.1 se presentan las funcionalidades probadas, en la sección 5.2 se detalla cómo fueron llevadas a cabo las sesiones, y en la sección 5.3 se presentan los incidentes encontrados.

5.1. Funcionalidades

A medida que se fueron identificando las misiones, las funcionalidades complejas se descompusieron en otras más simples, obteniéndose un inventario más detallado con mayor cantidad de funcionalidades respecto al inventario inicial. Se había planificado probar, mediante testing exploratorio, 465 funcionalidades; sin embargo, en la práctica se probaron 607.

5.2. Sesiones

Las sesiones se realizaban de forma individual, cada integrante mantenía una copia impresa de la misión asignada. Antes de comenzar con la sesión, se leía la misión, y de ser necesario se aclaraban las dudas con quien la había diseñado. A continuación, se fijaba el itinerario de la sesión y se procedía a su ejecución. Si se presentaban dudas de los resultados esperados mientras se ejecutaba la sesión, se consultaba a los integrantes del equipo que tenían más conocimiento de la aplicación, el manual de usuario existente o al cliente directamente.

El tiempo registrado en cada sesión incluía el de ejecución de las pruebas y el de registro en el sistema de seguimiento de los incidentes encontrados. La duración promedio de las sesiones dependió de la persona que ejecutaba la sesión. Uno de los miembros del equipo mantuvo sesiones de menos de 1 hora de duración en promedio, mientras que otro integrante mantuvo sesiones de 3 horas de duración en promedio.

En el proyecto se definieron 20 misiones para las cuales se realizaron un total de 40 sesiones entre los 6 miembros del equipo de pruebas.

En la tabla 1 se muestra la cantidad de sesiones ejecutadas por misión. Las misiones se muestran ordenadas por prioridad, siendo 1 la más alta. Estas prioridades fueron asignadas a partir del análisis de riesgo realizado en la etapa de planificación. A partir de la tabla, puede observarse que para la misión de mayor prioridad se ejecutaron 13 sesiones mientras que, para 2 misiones de prioridad 4 se ejecutaron 2 sesiones por cada una de las misiones.

Prioridad de las misiones	Cantidad de misiones	Cantidad de sesiones por misión
1	1	13
2	1	5
3	1	3
4	2	2
5	15	1

Tabla 1 – Cantidad de sesiones por misión

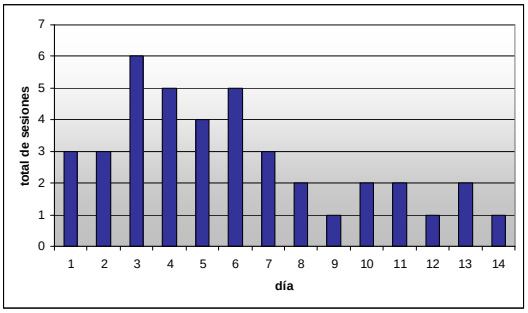


Figura 2-Total de sesiones por día

A pesar de existir una plantilla para registrar las sesiones, los documentos obtenidos diferían en contenido dependiendo del responsable de la sesión. Por ejemplo, algunos miembros del equipo describían de forma detallada sus pruebas, indicando para cada pantalla, todos los valores que le habían asignado a las entradas.

De la información registrada durante las sesiones se concluye que se invirtió un total de 100 horas para la ejecución de las sesiones. En la Figura 2 se muestra la cantidad de sesiones realizadas por día.

5.3 Incidentes

Durante el proyecto se encontraron un total de 120 incidentes. En la Figura 3 se observa que en el 25% de las 607 funcionalidades probadas se encontraron incidentes, o sea, se detectaron 154 funcionalidades con incidentes.

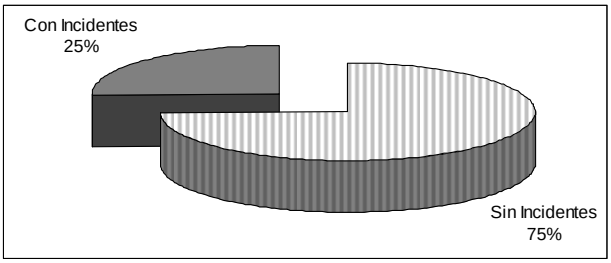


Figura 3- Cubrimiento de funcionalidades

El hecho de que la cantidad de funcionalidades con incidentes es mayor que la cantidad de incidentes encontrados se debe a que algunos incidentes fueron detectados en la ejecución de ciclos funcionales, con lo cual, un mismo incidente estaba involucrado con más de una funcionalidad.

Los incidentes fueron clasificados por prioridad: urgente, alta, normal o baja. A medida que iban siendo reportados, las prioridades asignadas eran validadas por el cliente usando el sistema de seguimiento de incidentes Mantis. En la Figura 4 se muestra los incidentes encontrados clasificados por prioridad.

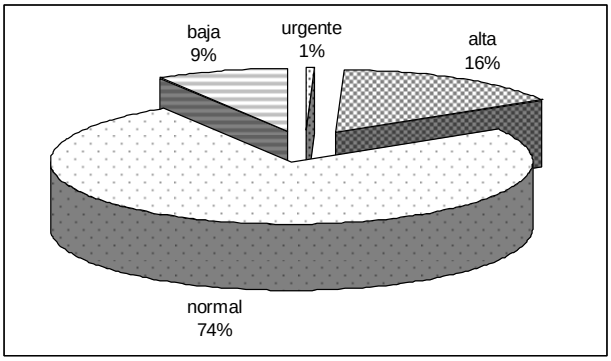


Figura 4- Incidentes por prioridad

Si bien había testers que inicialmente no estaban familiarizados con la aplicación y además se contaba con poco tiempo, se logró detectar un 16% de incidentes de prioridad alta y un 74% de prioridad normal. El cliente se mostró conforme con la calidad de los incidentes encontrados. Si bien la mayoría se debieron al cambio de plataforma, se detectaron incidentes funcionales que no estaban relacionados con dicha migración, y que no eran conocidos por el cliente. Esto agregó valor a los resultados obtenidos en el proyecto, cumpliéndose el objetivo de encontrar errores importantes en un corto período de tiempo.

6. Conclusiones

Se presentan en esta sección los aspectos positivos a destacar y los aspectos a mejorar en base a los resultados obtenidos en el proyecto presentado.

6.1 Aspectos positivos a destacar

El primer aspecto a destacar es la satisfacción del cliente con el proyecto en su conjunto, que fue considerado totalmente exitoso. El cliente consideró relevantes muchos de los incidentes encontrados y se mostró conforme con el cubrimiento de funcionalidades alcanzado.

Otro aspecto positivo es que se cumplió con la planificación del proyecto. El equipo de testing logró superar obstáculos tales como el desconocimiento de la aplicación a probar de algunos de sus miembros y la corta duración del proyecto. La primera conclusión es que hubiera sido imposible cumplir con un proyecto de estas dimensiones, crítico para el negocio y en un plazo tan exiguo, si no se hubiera aplicado testing exploratorio.

Respecto a la estrategia seguida durante las pruebas, resultó una buena práctica guiar las misiones en base a los resultados que se iban obteniendo, porque reforzó el análisis de riesgo realizado en la etapa de planificación del proyecto, poniendo el énfasis en los ciclos funcionales más críticos, con mayor cantidad de incidentes o menos probados.

Los informes de avance diarios generados y la información disponible en el Mantis, permitieron al cliente tener visibilidad del avance del proyecto en todo momento, y además tener conocimiento de los módulos y/o funcionalidades donde se concentraban los incidentes. A medida que los incidentes eran reportados los desarrolladores los iban solucionando para la versión siguiente, permitiendo comenzar con el ciclo de regresión inmediatamente después de culminar el ciclo de prueba 1. Cabe destacar que la gestión del testing exploratorio basado en sesiones permitió una comunicación fluida y permanente con el cliente.

6.2 Aspectos a mejorar

Se describen a continuación algunos de los aspectos a mejorar en la forma de trabajo.

Se plantea unificar los criterios de redacción de las sesiones, de forma análoga a lo logrado para el reporte de incidentes, que contribuyó a la buena comunicación con el equipo de desarrollo del cliente. Si bien se definió una plantilla para las sesiones, los testers la trabajaron en forma heterogénea, lo cual no facilita su lectura por parte del cliente.

Otra área de mejora es la utilización de herramientas de automatización de las sesiones para su reproducción en los ciclos de regresión. De esta forma, podría aumentarse la productividad de los testers experimentados que podrían dedicarse a nuevas funcionalidades o abordar las mismas con nuevas misiones.

La construcción de una herramienta que automatice el procesamiento de la información reportada en las sesiones es también un objetivo importante desde el punto de vista gerencial, ya que la recopilación de la información en forma manual llevó un tiempo considerable del proyecto.

Como resultado de todas las mejoras anteriores combinadas, se obtendrían métricas y mediciones más precisas, como por ejemplo, la cantidad de incidentes detectados por sesión, el tiempo relativo de preparación y manejador de base de datos y a otra plataforma.ejecución de las pruebas en cada sesión.

7. Referencias

[1] J. Bach, "Exploratory Testing Explained", The Test Practicioner, 2002, http://www.satisfice.com/articles/et-article.pdf

[2] C. Kaner, J. Falk, H. Nguyen, "Testing Computer Software, 2nd Edition", ISBN: 0471358460, Editorial Wiley, 1999.

[3] J. Bach, "What is Exploratory Testing? And How it differs from Scripted Testing", StickyMinds, Enero 2001.

[4] R. Black, "Managing the Testing Process, 2nd Edition". ISBN 0-471-22398-0, Editorial Wiley, 2002.

[5] Juha Itkonen and Kristian Rautiainen, "Exploratory Testing: A Multiple Case Study", Helsinki University of Technology, Software Business and Engineering Institute, pp 84-93, IEEE, 2005.

[6] J. Bach, "Session Based Test Management", Software Testing and Quality Engineering Magazine Vol.2, No. 6, Noviembre 2000, http://www.satisfice.com/articles/sbtm.pdf.

[7] Centro de Ensayos de Software (CES), http://www.ces.com.uy, 2006.

[8] B. Pérez, "Proceso de Testing Funcional Independiente (ProTest)", Tesis de Maestría en Informática, PEDECIBA Informática, Instituto de computación, Facultad de Ingeniería, Universidad de la Republica, Uruguay, ISSN: 0797-6410 - 06-11, 2006.

[9] E. Kit, "Software Testing In The Real World: Improving The Process", ISBN 0201877562, Addison Wesley, 1995.

[10] Mantis, http://www.mantisbt.org/index.php, 2006.

Agradecimientos

El presente trabajo ha sido desarrollado en el marco del proyecto COMPETISOFT (Mejora de Procesos para Fomentar la Competitividad de la Pequeña y Mediana Industria de software de Iberoamérica) del programa CYTED (Ciencia y Tecnología para el Desarrollo)

Una propuesta para la Elicitación de Requerimientos de Seguridad Basada en Preguntas

Vianca Vega Z.[1], Gloria Gasca H.[2], Edmundo Tovar C.[2], José Carrillo V.[2]
[1]*Universidad Católica del Norte,* [2]*Universidad Politécnica de Madrid*
vvega@ucn.cl, glogasca@yahoo.com, etovar@fi.upm.es, jcarrillo@fi.upm.es

Abstract

En el presente artículo se presenta una propuesta para la elicitación de requerimientos de seguridad, desarrollada a partir del método de análisis de requerimientos basado en preguntas, y con fundamento en algunas taxonomías de requerimientos de seguridad y en el modelo de madurez para la Ingeniería de Seguridad de Sistemas SSE-CMM.

La propuesta funciona a través de la incorporación de ciertos cuestionarios y formularios definidos a partir de los fundamentos ya mencionados. Este nuevo método es una primera aproximación para que las organizaciones desarrolladoras de software comiencen a incorporar la elicitación de requerimientos de seguridad en sus procesos.

1. Introducción

En los últimos años, la Ingeniería de Requerimientos ha alcanzado relevancia en el proceso de desarrollo de software, luego que los desarrolladores han tomado conciencia de lo importante que es definir de manera formal, correcta, y fácilmente modificable los requerimientos. La experiencia muestra que los errores cometidos durante el desarrollo de software, mientras más tarde se detecten, resultan más costosos, de allí la importancia de la incorporación de técnicas sistemáticas y repetibles desde las etapas tempranas de los procesos de desarrollo, especialmente aquellas relacionadas con la elicitación, validación y trazabilidad de los requerimientos.

Además de lo anterior, los desarrolladores de software también se han dado cuenta que en relación a la seguridad, no es suficiente la tendencia actual, según la cual, sólo una vez que se ha finalizado el desarrollo del software, busca proteger este producto a través de infraestructura montada sobre la plataforma en que funcionará. Esta estrategia, que es más bien reactiva, no ha sido suficiente, y esto se ve demostrado por la innumerable cantidad de ataques exitosos a los sistemas informáticos.

Por otro lado, también se ha determinado que la seguridad, por ser una propiedad emergente de los sistemas, es decir, que requiere el funcionamiento de varios componentes para que sea satisfecha, debe ser incorporada desde las etapas tempranas del desarrollo de software. De aquí la importancia de la presente investigación, cuyo objetivo es presentar una adaptación al método de análisis de requerimientos basado en preguntas [1], con el fin de analizar e incorporar los requerimientos de seguridad desde el inicio de los proyectos de desarrollo y su posterior trazabilidad e implementación, en cada etapa del ciclo de vida del software.

La adaptación propuesta se basa en algunas taxonomías y definiciones de requerimientos de seguridad presentadas por diversos autores [2-5]. El método se estructura de la siguiente forma: en base a taxonomías de requerimientos de seguridad y objetivos de seguridad, se determinan un conjunto de preguntas que ayudarán en la elicitación de las necesidades de seguridad del nuevo producto en desarrollo; se determina mediante una matriz de control de acceso la visibilidad y alcance de cada usuario del sistema en relación a las funcionalidades del mismo; luego se priorizan los requerimientos de seguridad determinados.

El presente artículo ha sido organizado de la siguiente forma: En la sección dos se presentan las taxonomías y definiciones utilizadas como fundamentos; En la tercera parte, se presenta brevemente el método de análisis de requerimientos basado en preguntas, para luego en la cuarta sección introducir la nueva propuesta. El artículo finaliza con la presentación de las conclusiones obtenidas y el trabajo futuros a realizar.

2. Fundamentos

En la presente sección se dan los fundamentos que permitieron el desarrollo de la propuesta para la elicitación de requerimientos de seguridad basada en preguntas. Además se parte presentando algunos conceptos asociados a la seguridad del software, a modo de justificación del trabajo realizado y la propuesta presentada.

2.1 Seguridad del Software

Mc Graw plantea que la seguridad del software es una idea de la Ingeniería de Software, que busca que un producto desarrollado continúe funcionando correctamente ante ataques maliciosos [6], es la construcción de software que puede resistir pro activamente los ataques.

Por otra parte, Viega y Mc Graw, en su libro "Building Security Software" [7], plantean que la seguridad puede ser vista como una medida de qué tan robusto es un sistema de software, respecto a una política de seguridad. A su vez, definen una política de seguridad como una regla de acceso a los recursos del sistema.

Los mismos autores agregan además la visión que la seguridad no es un aspecto que pueda ser incorporado a un sistema en cualquier instante, esto implica que al diseñar un producto de software, este diseño debe realizarse pensando en el aspecto de seguridad, dado que una vez desarrollado el producto, no será posible incorporarla exitosamente.

McGraw en su libro Software Security: Building Security In [8], plantea además que la seguridad del software se basa en tres pilares fundamentales: la administración del riesgo, la aplicación de prácticas específicas en etapas del ciclo de vida de desarrollo (mejores prácticas) y el conocimiento.

Entre estas prácticas que deben ser aplicadas durante el ciclo de desarrollo del software, tienen gran relevancia aquellas relacionadas con la etapa de requerimientos, en donde se deben especificar los Requerimientos de Seguridad.

Al igual que para los Requerimientos Funcionales se utilizan herramientas como los casos de uso, mediante los cuales se pretende obtener todas y cada una de las necesidades del cliente, en el caso de los requerimientos de seguridad, es importante tener en cuenta los requerimientos asociados a las funcionalidades de seguridad que el cliente manifiesta, por ejemplo, el uso de técnicas criptográficas como protección de su información, pero también es necesario tener en cuenta las propiedades emergentes de la seguridad. Para conseguir este objetivo, se plantea utilizar herramientas como los casos de abuso [3].

El método presentado en el presente artículo, es una propuesta para ayudar en la determinación de estos requerimientos de seguridad.

2.2 Taxonomía de Requerimientos de Seguridad

Las clasificaciones presentadas en esta sección, fueron utilizadas para la creación del cuestionario básico que ayuda en la identificación de los requerimientos de seguridad del nuevo sistema en desarrollo. Toma como base un conjunto de requerimientos mínimos y que son comunes a la mayoría de los productos de software que deben ser incorporados en los nuevos sistemas.

Firesmith presenta una taxonomía de requerimientos de seguridad [2], en la cual se identifican cuatro clasificaciones para los mismos, las cuales se presentan a continuación.

Requerimientos de seguridad puros. Dado que la seguridad es un atributo de calidad, los requerimientos de seguridad deberían consistir de un criterio de seguridad junto a un umbral mínimo requerido de una medida apropiada. Se identifican dos subtipos de requerimientos: el tipo Problema de defensa y el tipo Solución de defensa.

Requerimientos significativos de seguridad. Son funcionalidades del sistema que tienen ramificaciones de seguridad.

Requerimientos de seguridad del sistema. Permiten afianzar adecuadamente la seguridad del sistema. Algunos ejemplos de este tipo son el control de acceso y los antivirus.

Restricciones de seguridad. Son políticas obligatorias o medidas de contención para asegurar que se implementen los requerimientos de seguridad. Ejemplos de este tipo son: Identificación y autenticación; Integridad y no repudiación.

Los requerimientos de seguridad se refieren a conceptos tales como:

Subsistemas de Control de Acceso: Responsable de proveer la identificación, autenticación y autorización.

Subsistemas de Encriptación: Responsables de asegurar aspectos como la confidencialidad, no-repudiación e integridad de los mensajes.

Paquetes de Antivirus: Responsables de establecer los niveles necesarios para asegurar la integridad del software.

Las restricciones de seguridad se relacionan con:

Identificación y Autenticación

Encriptación

Integridad y No-Repudiación

Por otro lado, Shreyas [4] indica que Bashir en su artículo *Securing Network Software Application: Introduction*, clasifica el concepto de seguridad en múltiples dimensiones, entre las que se encuentran:

Autentificación. Verificar la identificación de los usuarios, es decir, que quien se está conectando sea quien dice ser.

Control de Acceso. Regulación de los privilegios de los usuarios, que cada usuario pueda realizar sólo las tareas que le corresponden.

Auditoría. Capacidad de registrar los eventos que afectan al sistema, de forma tal, que posteriormente se pueda reconstruir los estados pasados del sistema.

Confidencialidad. Evitar que ciertas entidades no autorizadas puedan acceder a información confidencial.

Integridad. Que la información no pueda ser modificada mediante mecanismos no permitidos.

Disponibilidad. Asegurar que el sistema estará disponible cuando el usuario lo requiera.

No repudiación. Asegurar que alguna entidad que ha participado en una comunicación, no pueda negar su intervención en ella.

2.3 Objetivos de seguridad

Si bien es cierto que las taxonomías de requerimientos de seguridad se encuentran fuertemente relacionadas con los objetivos de seguridad que a continuación se presentan, en esta sección se muestran por separado con el fin de resaltar las consideraciones que se deben tener presente si se desea incorporar la seguridad como un concepto global en el desarrollo de software, es decir, como una propiedad emergente del nuevo producto en desarrollo.

A la hora de elicitar los requerimientos de seguridad, es importante tener en cuenta un conjunto de objetivos, definidos por Viega y Mc Graw [7]. En el método propuesto, estos objetivos fueron considerados y utilizados para determinar los niveles de análisis necesarios para asegurar la correcta determinación de los requerimientos de seguridad. También fueron utilizados para validar que el método ha incorporado todos los aspectos necesarios que deben ser cubiertos en la elicitación de los requerimientos de seguridad.

Los objetivos planteados por Viega y Mc Graw [7] son los siguientes:

Prevención. Siempre se debe anticipar a las posibles fallas de seguridad.

Trazabilidad y Auditoría. Una de las formas de recuperarse de un ataque es conocer quién fue el atacante, qué hizo y cuándo lo hizo.

Monitoreo. Observar y monitorear constantemente el software en funcionamiento, permite detectar intrusiones.

Privacidad y confidencialidad. Mantener en secreto y bien resguardada la información que maneja el sistema.

Multiniveles de seguridad. No toda la información requiere los mismos niveles de seguridad, pero manejar esta diferencia no es una tarea fácil.

Administrar el anonimato. Este punto puede ser visto de dos aspectos, en ocasiones es necesario que se mantenga el anonimato de los usuarios, pero en otras ocasiones es primordial asegurar que no pueda existir anonimato en las transacciones del sistema.

Autentificación. Este objetivo, junto a la confiabilidad y la integridad, son considerados los más importantes. La autentificación es un aspecto crucial para la seguridad, dado que permite identificar con certeza quién realiza qué en un sistema.

Integridad. Esta meta busca controlar los cambios realizados en la información.

2.4 Modelo de Madurez para la Ingeniería de Seguridad de Sistemas.

Otro de los fundamentos, que a la vez justifica el desarrollo de la propuesta presentada, está dado por el modelo SSE-CMM (Systems Security Engineering Capability Maturity Model) [5].

Los modelos de madurez, permiten a una organización determinar qué tan bien se encuentra realizando sus procesos, sirviendo de guía además para ir avanzando en la mejora continua de los mismos. Para el caso del método propuesto, SSE-CMM fue utilizado para verificar que esta propuesta efectivamente incorpora las tareas mínimas que permiten a las organizaciones desarrollar sus procesos en un marco de calidad.

Dentro del alcance de SSE-CMM, se encuentra la presentación de una serie de actividades para lograr el desarrollo de productos de software confiables, y alcanzar un ciclo de vida para sistemas seguros [5]. La propuesta se desarrolla, considerando que la ingeniería de seguridad no es una actividad que pueda desarrollarse en forma aislada de otras especialidades de la ingeniería, en especial de la ingeniería de sistemas y de software. El modelo se estructura en dos dimensiones: Dominios y Capacidades. Un dominio es un conjunto de prácticas básicas que definen la ingeniería de seguridad, mientras que una capacidad se refiere a las prácticas genéricas que determinan la administración del proceso e institucionalizan la capacidad.

Este modelo, incluye como área clave en el proceso, la especificación de las necesidades de seguridad, lo que involucra *"...definir la base para seguridad en el sistema a fin de encontrar todos los requerimientos de seguridad, legales, políticos y organizacionales*" [5]. Este objetivo se puede lograr mediante el desarrollo de las siguientes prácticas básicas:

Entender las necesidades de seguridad del cliente, teniendo en cuenta que éstas están influenciadas por los riesgos de seguridad que le afectan.

Identificar leyes, políticas y restricciones aplicables. Se deben considerar aspectos internos y externos a la organización.

Identificar el contexto de seguridad del sistema, esto dado que existen muchos factores externos que afectan la seguridad.

Capturar la visión de seguridad del sistema en operación, esto incluye la identificación de roles, responsabilidades, flujos de información, bienes, recursos y protección física.

Capturar metas de seguridad de alto nivel. Se deben definir los objetivos que deben ser alcanzados.

Definir requerimientos relacionados con la seguridad. Tener en cuenta que la seguridad debería, dentro de lo posible, evitar impactos sobre las funcionalidades y desempeño del sistema.

Obtener acuerdos sobre la seguridad. Los requerimientos de seguridad finales, deben ser una reflexión completa y consistente sobre políticas, leyes y necesidades de los clientes.

3. Análisis de Requerimientos Basado en Preguntas

El análisis de requerimientos basado en preguntas, es una metodología para la Ingeniería de Requerimientos propuesta por Potts, Takahashi y Antón [1], quienes proponen una estructura cíclica, a la que llaman "Inquiry Cycle", para administrar el establecimiento de los requerimientos de software. La figura 1 muestra la estructura propuesta.

Este modelo consta de tres etapas: la documentación de los requisitos, su discusión y su evolución.

A continuación se explican brevemente cada una de las etapas, según la orientación dada por sus autores, incluyendo además las propuestas de cómo el método puede ser adaptado para los requerimientos de seguridad.

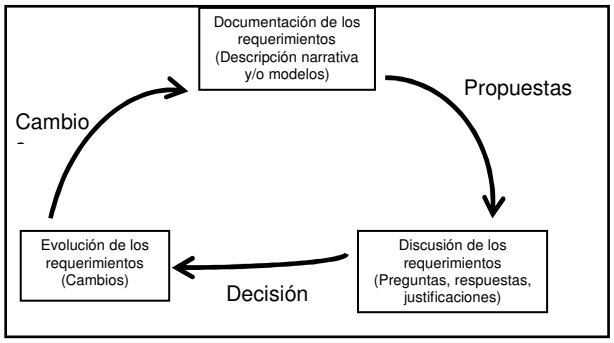


Figura 1. Técnica de educción de requerimientos "Inquiry-Based Requirement Analysis"

3.1. Documentación de los requerimientos

Los diferentes *stakeholders*[1] del sistema [9], escriben su propuesta de requerimientos. Estas propuestas pueden ser desarrolladas mediante descripciones narrativas, casos de uso u otros artefactos o simplemente como un punteo de los objetivos del sistema en desarrollo.

Aplicación al área de seguridad. Para el caso de requerimientos de seguridad, los *stakeholders* deberían incluir en su propuesta inicial sus apreciaciones en relación a la seguridad que debería incorporar el nuevo sistema. Tal como Potts, Takahashi y Antón recomiendan desarrollar casos de uso para los requerimientos funcionales, se pueden incorporar casos de abuso o casos de uso de seguridad.

3.2. Discusión de los requerimientos

Cada *stakeholder* revisa las propuestas publicadas y hace consultas o anotaciones respecto a ellas. Se produce una discusión en base a consultas, respuestas y justificaciones entre los mismos. Esta fase permite obtener un entendimiento detallado de cada uno de los requerimientos, y permite detectar si existen ambigüedades, restricciones u omisiones.

Aplicación al área de seguridad. Para reforzar la elicitación de requerimientos de seguridad, y asegurar que no queden sin considerarse aspectos importantes en este tema, esta fase del método puede potenciarse mediante la incorporación por parte de los ingenieros de requerimientos, de un conjunto de preguntas que cubran diversos aspectos de seguridad. Este conjunto de preguntas puede determinarse a partir de alguna taxonomía de requerimientos de seguridad, taxonomías

[1] **Stakeholders**: Persona u Organización que se ve afectada por el desarrollo del producto de software. Este término en ocasiones se traduce como "grupos de interés".

de ataques y amenazas o algún modelo de seguridad tal como SSE-CMM [5].

3.3. Evolución de los requerimientos

En base a la discusión generada en la fase anterior, se llega a un acuerdo entre los distintos *stakeholders*, permitiendo obtener una versión refinada de los requerimientos del sistema. Esta versión refinada, es el resultado de la modificación, eliminación o inclusión de algún requerimiento.

Aplicación al área de seguridad. En esta etapa se busca alcanzar el consenso entre los *stakeholders* en relación a las necesidades de seguridad del sistema. El obtener el acuerdo de las necesidades de seguridad es una de las práctica bases incluidas por SSE-CMM [5]. Si este acuerdo se logra, finaliza la aplicación de método, en caso contrario, se comienza un nuevo ciclo.

4. Propuesta para elicitar los Requerimientos de Seguridad

La propuesta para la elicitación de requerimientos de seguridad, toma como base las clasificaciones y definiciones multidimensionales de seguridad, presentadas en la sección dos.

El objetivo de la propuesta es presentar un método para elicitar los requerimientos de seguridad, en base a cuestionarios y formularios presentados a los clientes y usuarios del nuevo sistema a desarrollar, que incorporen los aspectos de seguridad de manera entendible para éstos, convirtiéndose en una herramienta que ayude además a la identificación de los posibles atacantes del sistema, y guíe a los usuarios a identificar sus necesidades de seguridad, que no siempre son concientes e intuitivos.

El método se desarrolla en base a un *Ciclo de Preguntas* similar al propuesto por Potts, Takahashi y Antón [1] (Ver sección 3) en donde se incluyen las siguientes actividades:

Determinación de los requerimientos de seguridad. Mediante los formularios y cuestionarios creados en base a las taxonomías y definiciones presentadas en la sección dos, se ayuda a los usuarios y clientes a tomar conciencia de sus necesidades de seguridad en forma individual. Para la aplicación de algunos de los formularios, se requiere que previamente se conozcan los requerimientos funcionales iniciales del sistema.

Documentación de los requerimientos de seguridad. En base a los resultados de la etapa anterior, se documentan mediante casos de abuso [10] o casos de uso de seguridad [11] las necesidades planteadas por los usuarios.

Discusión de los requerimientos. Esta etapa tiene el mismo objetivo que la etapa original planteada por Potts, Takahashi y Antón [1]. Se presentan a los usuarios y clientes las apreciaciones y necesidades determinadas de manera individual, para conducir de esta forma una discusión entre los *stakeholders* en base a consultas, respuestas y justificaciones.

Evolución de los requerimientos. Al igual que la etapa anterior, se mantiene el objetivo original de esta fase. Como resultado de la discusión, se obtiene una versión refinada de los requerimientos.

Los pasos del método propuesto se mantiene en desarrollo de manera iterativa hasta que los requerimientos se estabilicen producto del acuerdo de todos los *stakeholders*. La figura 2 muestra los pasos mencionados.

Si bien es cierto este método consta de las mismas etapas genéricas del proceso de Ingeniería de Requerimientos, la diferencia de esta propuesta radica en el tipo de artefactos que se aplican en cada una de las etapas, los cuales son específicos para la administración de requerimientos de seguridad.

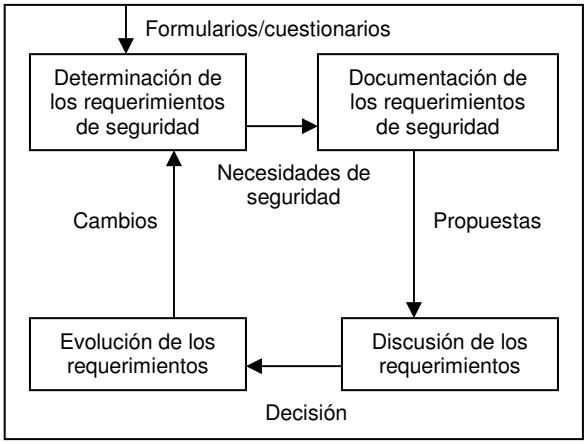


Figura 2. Método propuesto para la elicitación de requerimientos de seguridad

En la etapa de determinación de requerimientos de seguridad, se puede a la vez elicitar los requerimientos funcionales, utilizando entrevistas, cuestionarios u otra técnica conocida. En la etapa de documentación, además de la creación de casos de abuso o casos de uso de seguridad, los Ingenieros de requerimientos y analistas de sistemas pueden desarrollar los casos de uso habituales en los procesos de desarrollo actuales. La discusión y evolución de los requerimientos se desarrollan tanto sobre las características de seguridad, como las funcionalidades del sistema.

En este método propuesto, la esencia radical en la aplicación de cuestionarios que contengan las

preguntas adecuadas para permitir que los clientes y usuarios reconozcan sus propias necesidades de seguridad en relación al sistema y les ayude además a priorizarlas.

A continuación, las siguientes tablas muestran y justifican algunos de los cuestionarios y formularios que podrían ser utilizados en la primera fase de la propuesta cíclica.

En la tabla 1 se muestran algunos ejemplos de preguntas que se incluyen en el cuestionario inicial. Estas preguntas plantean al usuario objetivos de seguridad, sin utilizar términos técnicos que creen de su parte un rechazo.

Tabla 1. Ejemplos de preguntas incluidas en el cuestionario inicial.

Pregunta	Dimensión de seguridad
El objetivo de negocio al que este sistema dará soporte ¿Tiene alguna restricción gubernamental o legal?	Cumplimientos de leyes y regulaciones del negocio. (SSE-CMM [5])
¿Existe alguna funcionalidad del sistema que debe ser de acceso restringido? ¿Cuál?	Control de acceso (Requerimientos de Seguridad de la Taxonomía [2])
	Confidencialidad (Clasificación Bashir [4])
¿Para qué transacciones es importante registrar quién la desarrolló, modificó o eliminó?	Auditoría (Clasificación Bashir [4])
	Monitoreo (Objetivos de seguridad [7])
	No repudiación (Clasificación Bashir [4]).
¿Existe alguna información secreta que el sistema deba manejar?	Encriptación (Restricciones de seguridad [2])
	Confidencialidad. (Clasificación Bashir [4])
¿En qué horario el sistema debe estar disponible?	Disponibilidad (Clasificación Bashir [4])
¿Existe alguien que puede desear hacer daño al sistema u organización?	Reconocimiento de posibles atacantes. (Objetivo de Prevención [7])
¿Se compartirán o enviarán archivos entre usuarios del sistema?	Paquetes Antivirus (Requerimientos de Seguridad de la Taxonomía [2])

Tal como se mencionó, estas preguntas guían al usuario a pensar desde el inicio en las necesidades de seguridad en relación al sistema en desarrollo. En la tabla 1, la columna *Dimensión de seguridad* hace referencia a las definiciones y clasificaciones presentadas en la sección dos de este documento, que generó la consulta asociada.

La tabla 2 muestra el formulario que permite la determinación del control de acceso por parte de los usuarios del sistema, en relación a cada transacción o funcionalidad del mismo. Para la aplicación de este formulario, se requiere conocer previamente los requerimientos funcionales iniciales de los usuarios.

Tabla 2. Formulario de control de acceso

Roles de usuarios	**Transacción 1**	**……**	**Transacción n**
Rol 1	TA1		
Rol 2			TA2
……			
Rol m		TA3	

El control de acceso se determina mediante el uso de una matriz de accesos, en donde cada fila corresponde a un rol de usuario y cada columna representa una transacción o requerimiento de usuario. Se sugiere que en los ciclos iniciales de la aplicación del método se trabaje con requerimientos de usuarios, entendiendo por tal, la definición dada por Sommerville [12]: "*Los requerimientos del usuario son declaraciones, en lenguaje natural y en diagramas, de los servicios que se espera que el sistema provea y las restricciones bajo las cuales debe operar*".

A medida que avanzan los ciclos de aplicación del método, estos requerimientos de usuarios deberían irse refinando hasta convertirse en los requerimientos de sistema ("*Los requerimientos del sistema establecen con detalle los servicios y restricciones del mismo*" [12]), acordados y aceptados por todos los *stakeholders*.

El contenido de la matriz de control de acceso, define el tipo de acceso que cada rol de usuario tendría sobre cada uno de los requerimientos. En la tabla 2, a modo de ejemplo sólo aparecen tipos ficticios TA*i*. Se sugieren utilizar los siguientes tipos de acceso:

Consulta: El rol definido sólo puede acceder al requerimiento para realizar consultas sobre sus datos asociados.

Ingreso: El rol puede realizar el ingreso de nueva información sobre la funcionalidad indicada, pero no puede modificar información que existía previamente. El ingreso implica que además puede consultar.

Modificación: El rol puede ingresar nueva información, consultar y modificar información ingresada previamente. La modificación no considera eliminación de información.

Eliminación: El usuario asociado al rol indicado, tiene los privilegios de modificación definidos previamente, más la opción de eliminar información.

Sin acceso: El rol no puede acceder a la funcionalidad indicada.

Los tipos de acceso sugeridos, abarcan las clases más comunes, sin embargo, según la complejidad y sensibilidad de las funcionalidades del sistema en desarrollo, podrían incluirse otros tipos o mezclas de los presentados previamente.

La tabla 3 muestra un formulario cuyo objetivo es determinar la percepción de cada *stakeholders* en relación a la priorización de los requerimientos de seguridad.

En la tabla 3, las filas representan los distintos requerimientos de seguridad. Las columnas son los distintos requerimientos de usuarios o de sistema (aplicando el mismo criterio de la tabla 2 para determinar el control de acceso). Se solicita al usuario su prioridad por cada requerimiento, dado que como plantean Viega y McGraw [7], no todos los componentes de un sistema requieren los mismos niveles de seguridad (Objetivo de multiniveles de seguridad presentado en la sección dos).

La prioridad se representa de manera similar a lo sugerido por Robertson y Robertson [13] para el modelo de Ingeniería de Requerimientos Volere. Se le solicita al usuario asignar un número entre 1 al 5 a cada requerimiento, en donde los números representan la satisfacción/insatisfacción desde su perspectiva.

Por ejemplo, en términos de la satisfacción, por cada requerimiento, pedir al usuario poner una nota entre 1 y 5 donde 1 representa: "*Estaría absolutamente feliz si el requerimiento se implementa satisfactoriamente*" y 5 representa: "*Sería feliz si el requerimiento se implementa satisfactoriamente*". De manera similar para la insatisfacción, poner una nota de 1 a 5 donde 1 representa: "*Estaría ligeramente perturbado si no se implementa satisfactoriamente*" y 5 sería: "*Extremadamente molesto si no se implementa satisfactoriamente*".

Lo anterior implica que el usuario debe llenar dos formularios, uno para representar su satisfacción y otro para representar su insatisfacción.

Al igual que en el caso del formulario anterior, las dimensiones incluidas en la tabla 3 sólo son un conjunto básico de necesidades de seguridad típicas, las que podrían ser modificadas y refinadas según sea la complejidad y criticidad del sistema en desarrollo.

Tabla 3. Formulario priorización de requerimientos de seguridad

Dimensión de seguridad	**Transacción 1**	**……**	**Transacción n**
Me interesa que sólo accedan los usuarios autorizados. (Control de acceso)	5		4
Deseo que el usuario que realizó modificaciones no pueda negar su participación. (Audotoría)	3		1
Necesito realizar monitoreo de los cambios realizados y quién los realizó. (Monitoreo)	2		1
Quiero que la información asociada se almacene como clave. (Encriptación)	2		2
Requiero que la funcionalidad esté disponible cada vez que se necesite. (Disponibilidad)	5		5

5. Conclusiones

La propuesta presentada, es un método que puede ser adoptado y adaptado fácilmente por organizaciones desarrolladoras de software con poca experiencia en la incorporación de aspectos de seguridad en sus procesos, como un primer paso en la adopción de prácticas para el desarrollo de productos de software seguros.

Por otro lado, el método presentado también permite incorporar activamente a los usuarios del sistema en la tarea de determinar los requerimientos de seguridad desde las primeras etapas de desarrollo de

software. Uno de los aspectos distintivos de la propuesta, es que plantea los requerimientos de seguridad a los usuarios en un vocabulario sencillo y absolutamente entendible para ellos, evitando el uso de términos técnicos que puedan guiarlo a confusión.

La fortaleza del método propuesto, es que se basa en conceptos e investigaciones previas, integrándolas en una estrategia común.

Las siguientes etapas a realizar para los autores de la propuesta, es la validación de la elicitación de requerimientos de seguridad basada en preguntas, mediante su aplicación en proyectos de mediana escala. La idea de esta validación es que sea desarrollada en organizaciones que no poseen métodos ni estrategias formales para la incorporación de la seguridad desde etapas tempranas en el ciclo de desarrollo. La completitud y correctitud del método podrán ser validadas una vez que éste sea aplicado en casos de estudio reales. Actualmente se está dando inicio a la etapa de validación del método.

Tal como se mencionó en secciones previas, McGraw en su libro *Software Security: Building Security In* [8], plantea que la seguridad del software se basa en tres pilares fundamentales: la administración del riesgo, la aplicación de prácticas específicas en etapas del ciclo de vida de desarrollo (mejores prácticas) y el conocimiento. La propuesta presentada busca ser un aporte para el segundo y tercer pilar; apoya la incorporación de mejores prácticas al trabajar sobre la elicitación de requerimientos de seguridad, apoyados por el desarrollo de casos de abuso o casos de uso de seguridad. Por otro lado, se relaciona con el conocimiento, el cual involucra la captura, encapsulamiento y entrega del conocimiento de seguridad que puede ser usado para proveer fundamentos sólidos en prácticas de seguridad de software [8], dado que este método propone a través de sus preguntas incluidas en el cuestionario y a través de los formularios, los requerimientos típicos de seguridad que podrían incluirse en el nuevo producto en desarrollo.

Finalmente, en relación al modelo SSE-CMM, esta propuesta para la elicitación de requerimientos de seguridad, es un aporte para que las organizaciones desarrolladoras de software incorporen las prácticas bases para alcanzar mayores niveles de madurez en relación a la seguridad.

6. Referencias

[1] C. Potts, K. Takahashi, and A. Antón, "Inquiry-Based Requirements Analysis," in *IEEE Software*, 1994, pp. 21-32.

[2] D. Firesmith, "A Taxonomy of Security-Related Requirements," presented at Fifth International Workshop on Requirements for High Assurance Systems (RHAS`05 - Chicago) Chicago - USA, 2005.

[3] J. Mcdermott, "Abuse Case Models for Security Requirements Analysis.," presented at 17th Annual Computer Security Applications Conference New Orleans, Louisiana 2001.

[4] D. Shreyas, "Software Engineering for Security: Towards Architecting Secure Software.," 2002.

[5] "Systems Security Engineering Capability Maturity Model (SSE-CMM) Project," Junio 2003.

[6] G. McGraw, "Software Security," *IEEE Security & Privacity*, pp. 80-83, 2004.

[7] J. Viega and G. McGraw, *Building Secure Software: How to avoid security problems the right way*: Addison Wesley Profesional, 2001.

[8] G. McGraw, *Software Security. Building Security In*, 2006.

[9] H. Sharp, A. Finkelstein, and G. Galal, "Stakeholders Identification in the Requirements Engineering Process," presented at Tenth International Workshop on Database and Expert Systems Applications, 1999.

[10] G. Sindre and A. Opdahl, "Templates for Misuse Case Description," presented at Seventh International Workshop on Requirements Engineering: Foundation for Software Quality Interlaken, Switzerland, 2001.

[11] D. Firesmith, "Security Use Case," in *Journal of Object Technology*, vol. 2, 2003, pp. 53-64.

[12] I. Sommerville, *Ingeniería de Software*, 6ª Edición ed: Addison Wesley, 2002.

[13] S. Robertson and J. Robertson, *Mastering the Requirements Process*, 1999.

Um Processo de Engenharia de Requisitos Baseado em Reutilização de Ontologias e Padrões de Análise

Ricardo de Almeida Falbo, Aline Freitas Martins, Bruno Marques Segrini, Gleison Baiôco, Rodrigo Dal Moro
Departamento de Informática - Universidade Federal do Espírito Santo (UFES)
Av. Fernando Ferrari s/n, Campus de Goiabeiras–29.060-900– Vitória – ES – Brasil
falbo@inf.ufes.br, {alinefmart, bruno_segrini, gleison.baioco, rdalmoro}@yahoo.com.br

Julio Cesar Nardi
Centro Federal de Educação Tecnológica do Espírito Santo (CEFET-ES) – UnED/Colatina
Av. Arino Gomes Leal, 1700, 29.700-603–Colatina – ES - Brasil
julionardi@yahoo.com.br

Abstract

Software reuse is pointed as a key factor for quality and productivity improvement. Ideally, reuse should occur in all levels of abstraction, but higher levels of abstraction tend to lead to major benefits. In this paper, we present a Reuse-based Requirements Engineering process that focus on reusing ontologies and analysis patterns. Reports about its application are also discussed.

1. Introdução

A reutilização, em diversos níveis, tem sido apontada como um fator importante para o aumento da qualidade e da produtividade no desenvolvimento de software [1]. Dado que a Engenharia de Requisitos (ER) é um dos processos cruciais para o sucesso de um projeto [2], é muito importante que elementos relacionados a requisitos de software sejam reutilizados, visando a se obter esses benefícios.

Entretanto, surpreendentemente, a reutilização no processo de ER tem sido pouco praticada, ainda que haja diversos métodos de ER que, de alguma forma, enfatizam o reúso.

Este trabalho apresenta um processo de ER baseado na reutilização de ontologias e padrões de análise, voltado para o desenvolvimento de sistemas de informação, que procura empregar métodos e técnicas atualmente bastante adotados pela comunidade de desenvolvimento de software. Assim, espera-se vencer a inércia e efetivamente colocar em prática a reutilização na ER. A seção 2 aborda os temas Engenharia de Requisitos e Reutilização. A seção 3 apresenta o processo proposto. A seção 4 discute relatos de sua aplicação em um estudo de caso. Por fim, a seção 5 apresenta conclusões e perspectivas futuras do trabalho.

2. Engenharia de Requisitos e Reutilização

A Engenharia de Requisitos de Software é o ramo da Engenharia de Software que se preocupa com os objetivos do mundo real para as funções a serem desenvolvidas e restrições a serem levadas em conta no desenvolvimento de software [3].

Como o nome indica, o foco da ER está nos requisitos. Em relação ao tipo de informação que o requisito documenta, requisitos são classificados em funcionais (especificações de serviços que o sistema deve prover) e não funcionais (restrições impostas aos serviços ou funções oferecidos pelo sistema e ao processo de desenvolvimento) [4].

A ER é um processo complexo que envolve diversas atividades. Considerando o estado da prática, em um processo de ER convencional, geralmente, requisitos são inicialmente levantados utilizando-se técnicas como entrevistas, análise de documentos etc e documentados na forma de sentenças que descrevem as funções que o sistema deve prover (requisitos funcionais) e restrições a elas impostas (requisitos não funcionais). A seguir, modelos do sistema são

construídos, no que se convenciona chamar de Análise de Requisitos.

A modelagem é uma atividade central para o processo de ER. Modelos servem como uma base comum a todas as atividades desse processo. Por um lado, eles resultam das atividades de levantamento e análise de requisitos. Por outro, eles guiam esforços ulteriores de levantamento, análise, negociação e documentação [5]. Novamente levando-se em conta o estado da prática, os modelos mais comumente elaborados incluem diagramas de casos de uso e suas descrições, modelos conceituais da aplicação a ser desenvolvida (tais como diagramas de classes ou de entidades e relacionamentos, dependendo do paradigma adotado no desenvolvimento) e modelos comportamentais (tais como diagramas de estados e diagramas de interação).

Durante o levantamento e a análise de requisitos, os requisitos vão sendo documentados em um Documento de Especificação de Requisitos (DER) e por fim verificados e validados nas atividades de Verificação e Validação (V&V) de requisitos.

É necessário, ainda, gerenciar as mudanças nos requisitos acordados, manter uma trilha das mudanças e gerenciar os relacionamentos entre os requisitos e as dependências entre o DER e os demais artefatos produzidos no processo de software, atividades realizadas no âmbito da Gerência de Requisitos.

Analisando o processo de ER, é possível notar que a reutilização pode ser útil, sobretudo no reúso de requisitos de sistemas similares e de modelos (com destaque para os modelos conceituais). Contudo, na prática, na grande maioria das vezes, requisitos e modelos são construídos a partir do zero. Ou seja, requisitos iniciais são levantados junto aos interessados, modelos são construídos levando-se em conta esses requisitos iniciais, que são posteriormente refinados e novamente modelados, até se atingir um acordo com o cliente sobre o que o sistema deve prover. Entretanto, essa abordagem tem se mostrado insuficiente, pois desconsidera a reutilização do conhecimento já existente na organização acerca do domínio do problema.

A reutilização no desenvolvimento de software tem como objetivos principais melhorar o cumprimento de prazos, diminuir custos e obter produtos de maior qualidade [1]. Artefatos de software podem ser reutilizados, investindo-se esforço na sua adaptação a novas situações. Uma vez que esses itens tenham sido desenvolvidos para reúso, o esforço de adaptação e reutilização tende a ser menor.

No que concerne à ER, os principais esforços em direção à reutilização estão na Engenharia de Domínio [6] e no uso de padrões de análise [7] [8].

Padrões são guias. Eles são usados para descrever soluções de sucesso para problemas de software comuns, refletindo a estrutura conceitual dessas soluções [9]. Padrões de análise são geralmente definidos a partir de modelos conceituais de aplicações, sendo descobertos e não inventados [7]. Ou seja, a partir da análise de diversos eventos de negócio do mesmo tipo, um padrão de análise pode ser derivado por meio da abstração de elementos comuns [8].

A Engenharia de Domínio representa um enfoque sistemático para a produção de componentes reutilizáveis que engloba atividades de Análise, Projeto e Implementação de Domínio, as quais objetivam, respectivamente, representar requisitos comuns de uma família de aplicações por meio de modelos de domínio, disponibilizar modelos arquiteturais para aplicações a partir de um único modelo de domínio e disponibilizar implementações de componentes que representam funcionalidades básicas de aplicações relacionadas a um domínio [6].

Claramente a Análise de Domínio é a atividade diretamente ligada à reutilização na ER. Diversos são os métodos de Análise de Domínio existentes, dentre eles FODA, ODM, EDLC, FORM, RSEB e Catalysis [6]. Falbo et al. [10] propõem o uso de ontologias como modelos de domínio e apresentam um processo de Engenharia de Domínio baseado em ontologias. Neste contexto, uma ontologia é um artefato de engenharia, constituído de um vocabulário de termos organizados em uma taxonomia, suas definições e um conjunto de axiomas formais usados para criar novas relações ou para restringir as suas interpretações segundo um sentido pretendido [11].

Cima et al. [12] destacam que a Análise de Domínio pode ser considerada como uma atividade anterior ao desenvolvimento de software (e, portanto, anterior à ER), uma vez que esse se utiliza dos modelos daquela.

É inegável que a reutilização na ER vem acontecendo de alguma forma, sobretudo informalmente. Por exemplo, a partir de Documentos de Especificação de Requisitos de projetos anteriores (sobretudo os similares), restrições, listas de interessados, glossários e até requisitos podem ser reutilizados [8]. Apesar disso, parece que a inserção da prática de reutilização como parte integral do processo de ER, de fato, ainda não aconteceu. As principais razões para esse fato têm cunhos metodológicos, tecnológicos, culturais e econômicos [13].

Nuseibeh e Easterbrook [14], em seu mapa para o futuro da ER, colocaram o reúso de modelos como um dos maiores desafios da ER nos anos 2000. Eles acreditavam que, para muitos domínios de aplicação, modelos de referência para especificação de requisitos seriam desenvolvidos, de modo que o esforço de desenvolver modelos de requisitos a partir do zero fosse reduzido. De alguma forma, isso, de fato, vem ocorrendo, ainda que não propriamente na forma de modelos de referência, mas por meio da construção de diversas ontologias de domínio e pelo crescente número de padrões de análise disponíveis.

Dado o panorama atual, este trabalho apresenta um processo de Engenharia de Requisitos *com* reutilização baseado em ontologias e padrões de análise, desenvolvido para ser aplicado ao desenvolvimento de sistemas de informação. Esse processo procura empregar métodos e técnicas atualmente adotados em larga escala pela comunidade de desenvolvimento de software, visando a minimizar alguns dos fatores culturais e metodológicos que têm restringido a difusão da reutilização na ER.

3. Um Processo de Engenharia de Requisitos Baseado em Reutilização

A Figura 1 mostra as atividades do processo de ER proposto.

O processo começa com um levantamento preliminar de requisitos, cujo objetivo é definir o escopo do projeto e enumerar os principais requisitos funcionais e não-funcionais. Paralelamente, um modelo conceitual preliminar deve ser elaborado, tomando por base ontologias e padrões de análise relacionados ao domínio do problema. Como produto dessa atividade, um DER preliminar é produzido.

Com base nos requisitos iniciais e no modelo conceitual preliminar, um levantamento de requisitos detalhado é conduzido e o modelo conceitual da aplicação é refinado. Finalmente, modelos comportamentais da aplicação são construídos.

Paralelamente a esse fluxo de trabalho principal, a documentação dos requisitos vai sendo elaborada (DER Preliminar e DER), verificada, validada e gerenciada. Uma vez aprovada, passa-se para a solução técnica propriamente dita (projeto, implementação etc). Senão, há um retorno para a correspondente atividade, visando a corrigir os problemas detectados.

A seguir, cada uma dessas atividades é discutida com um mais de detalhes.

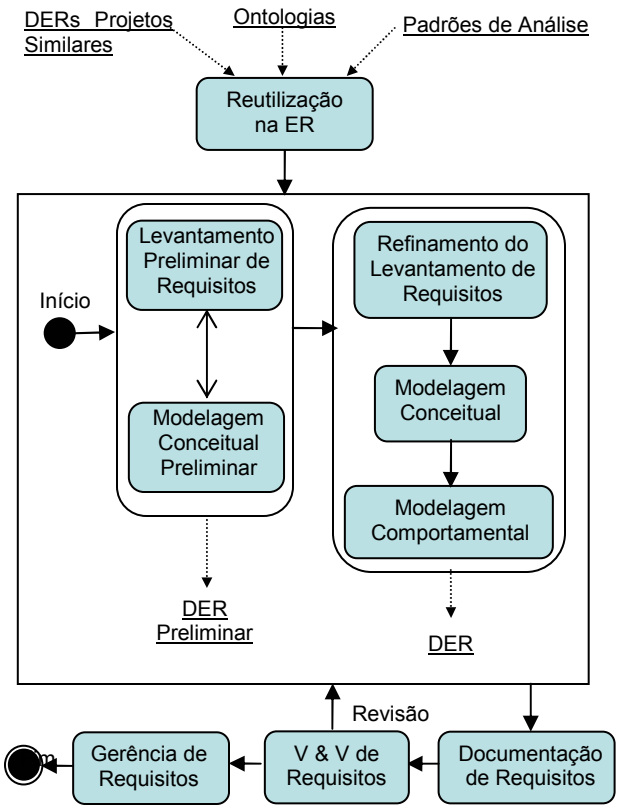


Figura 1 - Processo de Engenharia de Requisitos com reutilização

Levantamento Preliminar de Requisitos

O objetivo desta atividade é estabelecer o escopo do software a ser desenvolvido, especificando um conjunto de funcionalidades (requisitos funcionais) associado a outras características desejadas (requisitos não funcionais). Requisitos são inicialmente levantados e documentados na forma de sentenças descrevendo essas funções e restrições, como ilustra a Tabela 1. Tipicamente, diversas técnicas de levantamento de requisitos são empregadas, tais como entrevistas, investigação de documentos, observação, dinâmicas de grupo etc. Uma característica fundamental dessa atividade é o seu enfoque em uma visão do cliente/usuário. Em outras palavras, está-se olhando para o sistema a ser desenvolvido por uma perspectiva externa. Ainda não se está procurando definir a estrutura interna do sistema, mas sim procurando saber que funcionalidades o sistema deve oferecer ao usuário e que restrições elas devem satisfazer.

A reutilização nesta etapa tem um caráter mais informal. Recomenda-se a inspeção de DERs de projetos já concluídos, sobretudo aqueles similares, buscando-se reutilizar definições de requisitos, principalmente requisitos não-funcionais.

Modelagem Conceitual Preliminar

O objetivo desta atividade é definir um modelo conceitual preliminar para a aplicação a partir de ontologias e padrões de análise. A ênfase neste momento é a captura dos principais conceitos envolvidos, definindo uma estrutura interna básica do sistema, mas sem uma preocupação com a elaboração de um modelo completo, incluindo atributos e operações. Quando ontologias são reutilizadas, conceitos, relações e propriedades são mapeados, respectivamente, para classes, associações e atributos no modelo da aplicação [10]. No caso de padrões de análise, como esses são normalmente escritos na forma de diagramas convencionais (tais como diagramas de classes), eles podem ser reutilizados diretamente.

Alguns domínios podem ter vasto material disponível e, portanto, a seleção de ontologias e padrões de análise relevantes para o problema em questão é uma tarefa importante desta etapa. É útil examinar diversos modelos, procurando adaptá-los à realidade do projeto em questão, tomando por base os requisitos inicialmente levantados.

Como produto final desta etapa, um Documento de Especificação de Requisitos Preliminar é elaborado, basicamente contendo uma descrição do mini-mundo apresentando o problema e seu contexto, um conjunto de requisitos funcionais e não funcionais devidamente identificados, como ilustrado na Tabela 1, e um modelo conceitual preliminar.

Refinamento do Levantamento de Requisitos

De posse dos requisitos iniciais e do modelo preliminar, um refinamento dos requisitos deve ser realizado. As técnicas de levantamento de requisitos empregadas no levantamento preliminar de requisitos são igualmente válidas, sendo que outras, como a prototipagem e a modelagem de casos de uso, são indicadas.

No que se refere à modelagem de casos de uso, é interessante notar que casos de uso custodias (aqueles que lidam basicamente com a inclusão, exclusão, alteração e consulta de itens de informação) são mais facilmente identificados a partir do modelo conceitual preliminar, acelerando o processo. Outros, especialmente os relacionados ao negócio em si, ditos casos de uso essenciais, normalmente estão registrados de alguma forma nos requisitos funcionais iniciais. Assim, a maior parte do trabalho a ser feito neste momento está na identificação dos atores e na elaboração da descrição detalhada de cada caso de uso.

Modelagem Conceitual da Aplicação

Com os requisitos do sistema bem definidos, casos de uso especificados e um entendimento preliminar dos conceitos do domínio do problema estabelecido, é possível avançar para a proposição do modelo conceitual final da aplicação.

O modelo conceitual preliminar é, então, refinado. Tipicamente, atributos são adicionados e associações revisadas.

Atenção especial deve ser dada às multiplicidades, especialmente quando ontologias deram origem ao modelo conceitual preliminar, dado que, em geral, ontologias utilizam-se da idéia de compromissos ontológicos mínimos [15], ou seja, procuram dar uma visão mais abrangente acerca do domínio que modelam e tendem a impor menos restrições que uma aplicação específica.

Padrões de análise podem ser reutilizados, quando uma porção do modelo estiver sendo refinada e se detectar a ocorrência de um problema para o qual há um padrão de solução catalogado.

Além disso, novas classes e associações podem ser necessárias para tratar questões específicas da aplicação, descritas nos casos de uso. Um dicionário de dados do projeto deve ser elaborado e as definições dos termos da ontologia podem ser reutilizadas.

Modelagem Comportamental da Aplicação

Para finalizar a fase de modelagem de análise, assim como em qualquer processo de ER, modelos comportamentais da aplicação, tais como diagramas de estados e diagramas de interação, devem ser elaborados quando necessário.

Documentação de Requisitos

A documentação dos requisitos deve ocorrer ao longo de todo o processo de Engenharia de Requisitos. Dois documentos principais devem ser elaborados: o Documento Preliminar de Especificação de Requisitos, produto da primeira fase de levantamento e modelagem de requisitos, e o Documento de Especificação de Requisitos, que contém, além de uma descrição do problema e listas de requisitos, os diversos diagramas produzidos ao longo do processo de ER e um dicionário de dados do projeto.

Verificação e Validação de Requisitos

A atividade de Verificação e Validação (V&V) de requisitos se mantém praticamente inalterada em relação a um processo convencional de ER. A verificação se ocupa de checar se um documento de requisitos está sendo construído de forma correta, de acordo com padrões previamente definidos pela

organização, sem conter requisitos ambíguos, incompletos, incoerentes ou impossíveis de serem testados. A validação, por sua vez, avalia se os requisitos e modelos documentados atendem às reais necessidades e requisitos dos usuários / clientes.

Vale destacar a importância de se verificar se efetivamente as classes derivadas de conceitos de uma ontologia mantêm a mesma semântica em relação à conceituação por ela estabelecida, isto é, se os termos adotados são consistentes e se as restrições são respeitadas.

Gerência de Requisitos

Assim como a V&V, a gerência de requisitos se mantém com poucas alterações em relação a um processo convencional de ER. Deve-se destacar o fato de ser criado mais um nível de rastreabilidade em relação à origem de requisitos e de classes, uma vez que esses elementos podem ser derivados de ontologias e padrões de análise. Dessa maneira, devem-se manter ligações explícitas para esses artefatos.

4. Aplicação do Processo Proposto: Um Estudo de Caso

O processo proposto foi aplicado em um estudo de caso realizado com três grupos em uma disciplina no Mestrado em Informática da Universidade Federal do Espírito Santo. O problema proposto foi comum aos três grupos – alocação de recursos humanos a projetos em organizações de software – mas em cada grupo havia um representante de uma organização de software real e, assim, cada grupo tinha de levar em consideração os requisitos de um sistema para atender a essa organização específica.

Os seguintes artefatos foram disponibilizados para reúso: uma ontologia para modelagem de corporações [16], duas ontologias de organizações de software ([17] e [18]), o catálogo de padrões de análise de Fowler [7] e os padrões de análise definidos em [16].

Todos os três grupos seguiram o processo proposto, cabendo destacar que os correspondentes Documentos Preliminares de Especificação de Requisitos, nos quais são listados os requisitos funcionais e não funcionais da aplicação obtidos no Levantamento Preliminar de Requisitos, estavam bastante distintos uns dos outros.

No passo seguinte, Modelagem Conceitual Preliminar, todos os grupos reutilizaram preferencialmente ontologias. Os grupos 1 e 2 focaram basicamente na ontologia para modelagem de corporações proposta por Cota [16], enquanto o grupo 3 preferiu construir seu modelo conceitual preliminar com base na ontologia de organizações de software proposta em [18]. Esse terceiro grupo utilizou também o padrão de análise de alocação de recursos proposto em [16].

Em todos os casos, a confecção do modelo conceitual preliminar consistiu da extração de partes de uma ontologia com a qual a aplicação mais se comprometia em relação à conceituação estabelecida, sendo feitos pequenos ajustes para adequação aos requisitos específicos da aplicação, algumas vezes usando outras ontologias e padrões de análise (como no caso do grupo 3).

Perguntados sobre a preferência pelo reúso de ontologias, todos os grupos foram enfáticos ao afirmar que as ontologias eram bastante abrangentes, davam uma visão geral do domínio e estabeleciam um vocabulário bem definido a ser usado, o que permitiu facilmente extrair um modelo preliminar de classes. A ontologia de organizações de software proposta em [17] foi preterida em relação às demais, exatamente por sua apresentação ser muito fragmentada. O mesmo foi dito em relação aos padrões de análise, ainda que todos os grupos tenham apontado que a sua análise tenha ajudado de alguma forma na concepção dos respectivos modelos conceituais preliminares e, em alguns casos, mais aplicados na modelagem conceitual final da aplicação. Ainda no que diz respeito aos padrões de análise, em relação aos propostos por Fowler [7], houve comentários apontando que o vocabulário utilizado não era muito intuitivo, o que dificultava a sua aplicação. Esse problema não foi sentido em relação aos padrões de análise propostos em [16], tendo em vista que a autora utiliza os mesmos termos empregados em sua ontologia para definir os padrões de análise.

Em relação ao Refinamento do Levantamento de Requisitos, o fato de haver disponíveis um Documento Preliminar de Especificação de Requisitos e um modelo conceitual preliminar foi apontado como útil para a identificação de casos de uso. Contudo, em relação à elaboração das descrições de casos de uso, foi relatado que praticamente não houve influência da abordagem no esforço despendido. Ou seja, o processo correu como um processo convencional, sem reutilização.

Na etapa de Modelagem Conceitual da Aplicação, segundo os relatos dos três grupos, o trabalho foi bastante simplificado, com destaque para a elaboração do dicionário de dados do projeto. O trabalho foi concentrado em ajustes de multiplicidades de associações e na definição de atributos. Alguns dos grupos, inclusive, voltaram a inspecionar as ontologias e padrões de análise, sendo que dois grupos foram buscar nos padrões de análise atributos para as classes.

Novas classes e associações foram detectadas por dois grupos (uma classe e uma associação em um deles, apenas uma classe em outro). Por fim, dado que todos os grupos se apoiaram substancialmente em ontologias para a construção de seus modelos conceituais, a elaboração dos respectivos dicionários de dados foi conduzida, basicamente, reutilizando-se definições de termos das ontologias.

Deste ponto do desenvolvimento em diante, tendo em vista que o processo segue uma abordagem bastante próxima da convencional, pouco há a relatar. Cabe registrar apenas um comentário importante acerca da verificação. O grupo 3 relatou que, durante a verificação, foi detectado que alguns conceitos estavam sendo utilizados com uma semântica diferente da proposta pela ontologia. Isso decorria de um exame superficial da ontologia (apenas o modelo gráfico estava sendo analisado em um primeiro momento).

A seguir, um relato mais detalhado do trabalho realizado pelo grupo 3 é apresentado, focando nas atividades de Levantamento Preliminar de Requisitos, Modelagem Conceitual Preliminar, Refinamento do Levantamento de Requisitos e Modelagem Conceitual da Aplicação. Um breve comentário é feito, também, sobre a Verificação e Validação de Requisitos (V&V) e sobre a Gerência de Requisitos, apresentando parcialmente o relatório de rastreabilidade do projeto.

4.1. Estudo de Caso Detalhado

O grupo 3 elaborou a especificação de requisitos para o problema da alocação de recursos no contexto da Acropolis Soluções em Tecnologia da Informação, uma organização de desenvolvimento de software que conta com aproximadamente 50 colaboradores.

A Acropolis possui diferentes equipes para atender seus clientes, sendo o principal deles, uma companhia siderúrgica. Para essa companhia siderúrgica, há uma equipe de desenvolvimento de novos sistemas e outras duas equipes responsáveis por manutenções para áreas específicas (Sistemas Administrativos e Sistemas de Produção). Membros de uma equipe podem ser alocados para projetos de outras equipes. Dessa forma, gerentes de projeto precisam conhecer a alocação dos recursos nos vários projetos.

Levantamento Preliminar de Requisitos

O levantamento preliminar de requisitos do sistema proposto foi feito com base em entrevistas com os gerentes das equipes, tendo sido identificados requisitos funcionais e não-funcionais, parcialmente descritos na Tabela 1. Vale ressaltar que o sistema é responsável apenas pela alocação dos colaboradores. O cadastro de colaboradores e o gerenciamento dos processos são realizados por outros sistemas com os quais o sistema proposto deve ser integrado (RNF02).

Tabela 1 – Tabela Parcial de Requisitos.

RF001 - O sistema deve apoiar a alocação de colaboradores a projetos de acordo com a equipe em que cada colaborador se encontra.
RF002 - O sistema deve permitir que o gerente de uma equipe solicite a alocação de um colaborador de outra equipe.
RF003 - Um colaborador deve ser alocado com base em suas competências e papéis que pode assumir, e as alocações devem ser feitas para atividades de um projeto.
RNF01 - O sistema deve rodar em plataforma *Web*.
RNF02 - O sistema necessita de uma interface com os sistemas de cadastro de colaboradores e de controle de processos já existentes.
RNF03 - O sistema deve garantir as informações de seus usuários, controlando o acesso ao mesmo.

Modelagem Conceitual Preliminar

Levantados os requisitos iniciais do sistema, partiu-se para a inspeção dos artefatos disponíveis para reúso. O modelo de classes preliminar do sistema foi desenvolvido, baseando-se, sobretudo, na ontologia organizacional descrita em [18] e utilizando o padrão de alocação de recursos proposto em [16]. A Figura 2 apresenta parte da ontologia organizacional usada, enquanto a Figura 3 apresenta o modelo conceitual preliminar derivado. Vale destacar que essa ontologia organizacional foi desenvolvida de forma integrada a uma ontologia de processo de software [19]. Assim, os conceitos da primeira estão destacados.

Como se pode notar a partir dos modelos mostrados nas figuras 2 e 3, as classes `Atividade`, `Competencia`, `Cargo`, `Projeto`, `Equipe`, `Pessoa` e `Acumulo`, e as respectivas associações, tiveram origem nos conceitos homônimos da ontologia proposta em [18]. Diversos conceitos e relações foram desconsiderados, dado que não eram relevantes para o sistema em questão, tais como `Organização` e `Unidade Organizacional` e suas respectivas relações. No que diz respeito à hierarquia de `Processo de Software` apenas uma classe para tratar processos de projeto era relevante para o sistema em questão, dando origem à classe `Processo`. Por fim, identificou-se a necessidade da classe associativa `Alocacao` a partir da inspeção do padrão de análise "Alocação de Recursos" proposto em [16].

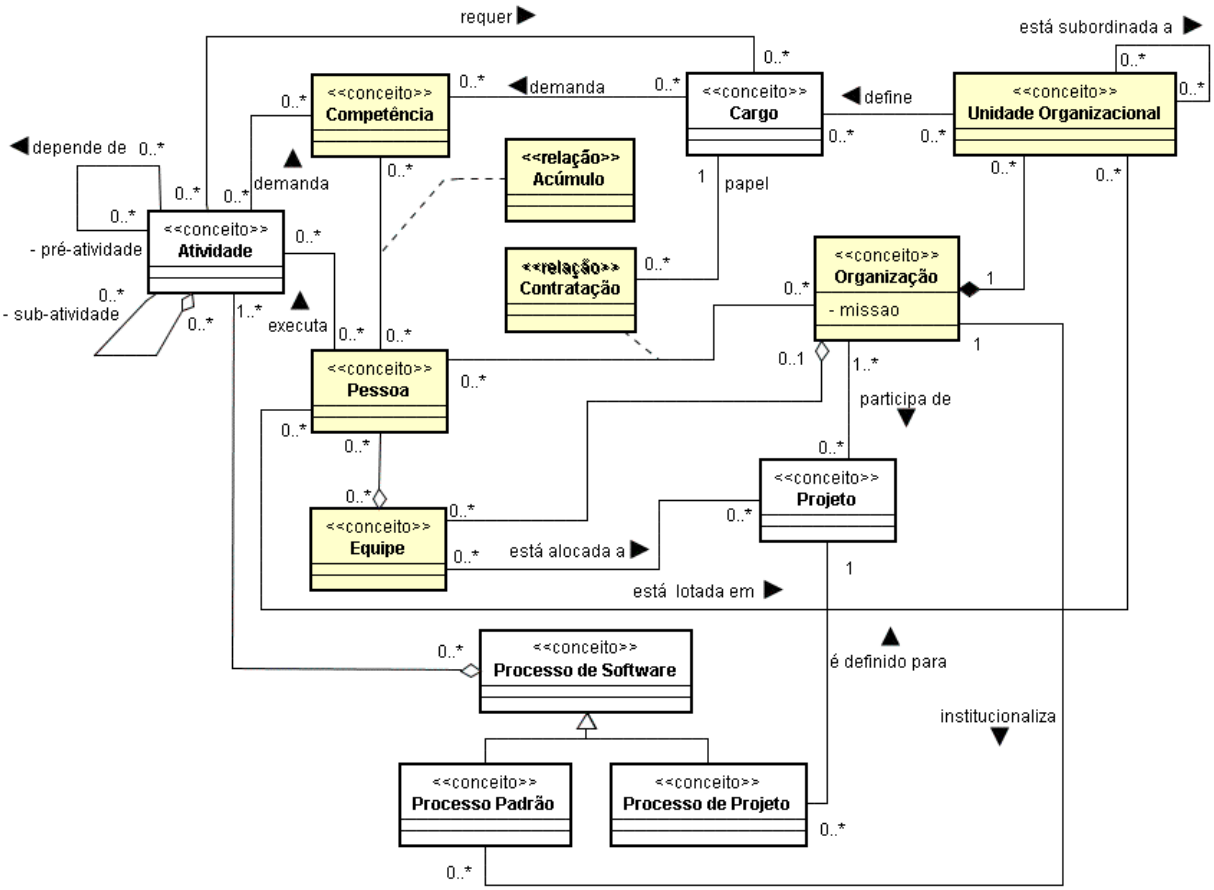


Figura 2 - Ontologia de Organizações de Software – Estrutura Organizacional [18]

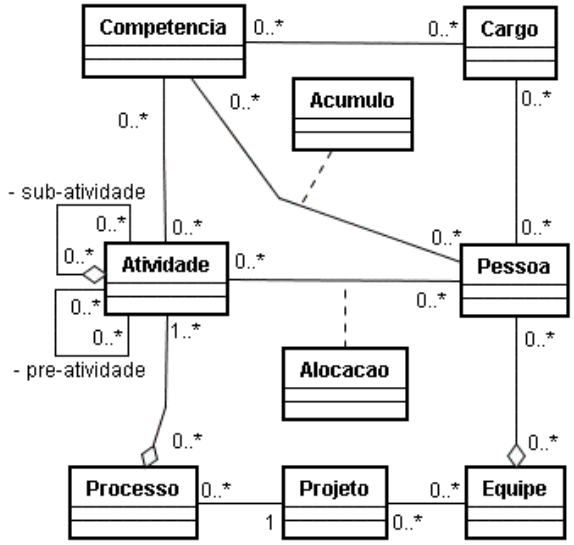


Figura 3 – Modelo conceitual preliminar

A maior parte das classes e associações foi derivada da ontologia proposta em [18], pois seus conceitos tinham definições e relações muito semelhantes aos identificados no contexto da Acropolis. Havia, ainda, similaridades com os modelos dos sistemas de cadastro de colaboradores e de gerência de projetos já existentes, aos quais o sistema proposto deverá ser integrado.

Refinamento do Levantamento de Requisitos

Com os requisitos levantados e definido um modelo preliminar de classes, a elaboração do modelo de casos de uso foi facilitada. As principais funcionalidades do sistema já estavam identificadas na lista de requisitos funcionais (Tabela 1). Além disso, o modelo de classes preliminar guiou a identificação de casos de uso custodiais. Assim, bastou identificar os atores e fazer a descrição dos casos de uso. A Figura 4 mostra um diagrama de casos de uso parcial do sistema proposto.

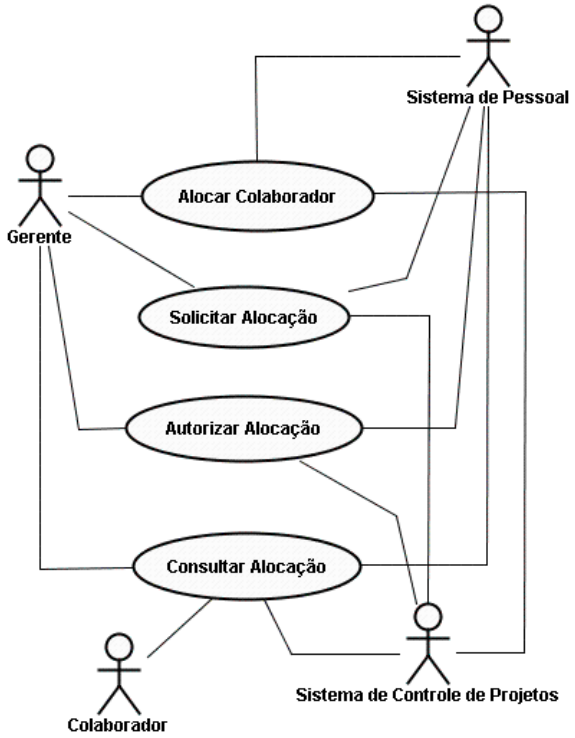


Figura 4 – Diagrama de Casos de Uso

Modelagem Conceitual da Aplicação

De posse do modelo preliminar, foi realizada uma análise mais detalhada dos requisitos, tomando por base as descrições dos casos de uso, e algumas alterações nesse modelo foram necessárias, dando origem ao modelo da Figura 5.

Confrontado requisitos e o modelo preliminar, identificou-se a necessidade de incluir mais uma classe, `Usuario`, para atender ao requisito RNF003.

Atributos foram adicionados às diversas classes, sendo que alguns deles estavam descritos na ontologia e no padrão utilizados como base, no caso deste último, os atributos da classe associativa `Alocacao`.

Algumas multiplicidades tiveram de ser alteradas para ficar em linha com as regras de negócio da Acropolis. Dado que na organização em questão cada projeto tem apenas uma única equipe associada, a multiplicidade entre `Projeto` e `Equipe` foi alterada na extremidade próxima à `Equipe`. Situações análogas ocorreram nas associações de agregação entre atividades (`sub-atividade`) e entre atividades e processos. Na Acropolis, uma atividade só pode ser parte de, no máximo, uma outra atividade ou processo, sendo que, neste último caso, toda atividade é parte de um processo. Além disso, ao se montar uma equipe, pelo menos um colaborador deve ser informado, o que fez com que fosse alterada a multiplicidade da associação entre `Pessoa` e `Equipe` na extremidade próxima à `Pessoa`.

Por fim, para atender aos casos de uso relacionados à definição de alocações, cujo ator é o gerente de projeto, foi necessária a identificação do papel *gerente* de uma equipe. Para tal, houve a inclusão de uma nova associação entre as classes `Pessoa` e `Equipe`, pela qual é indicado quem é o gerente de uma determinada equipe.

Vale ressaltar, ainda, que a construção do dicionário de dados do projeto foi facilitada, pois grande parte das definições dos termos da ontologia foi diretamente utilizada para descrever classes e atributos.

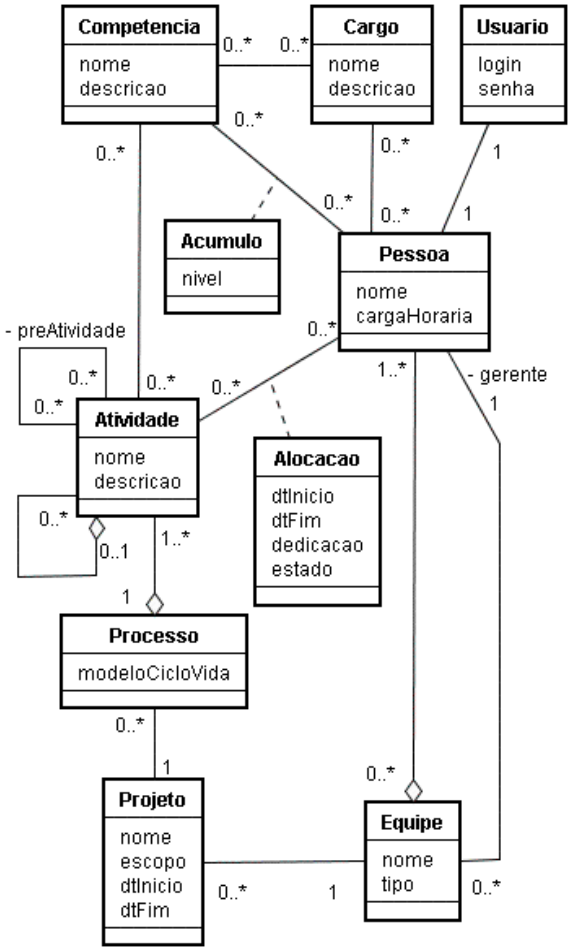


Figura 5 – Modelo conceitual da aplicação

V&V e Gerência de Requisitos

Em relação à Verificação e Validação (V&V) de Requisitos, foi muito importante verificar se efetivamente as classes derivadas de conceitos da ontologia mantinham a mesma semântica em relação à conceituação estabelecida por esta última e se os

termos adotados eram consistentes. Por ocasião da V&V do modelo preliminar de requisitos, detectou-se, por exemplo, que havia má interpretação de alguns conceitos e mudanças no modelo foram realizadas.

No que se refere à Gerência de Requisitos, mais especificamente em relação à rastreabilidade, foram adicionadas informações sobre a origem das classes do modelo de análise.

Em uma abordagem convencional, um relatório de rastreabilidade, normalmente, registra, para cada requisito, requisitos dependentes, conflitantes, casos de uso e pacotes ou classes necessários para tratar o requisito. Seguindo a abordagem de reúso adotada, foi informada, ainda, a origem de uma determinada classe, quando pertinente. A Figura 6 mostra parte do Relatório de Rastreabilidade gerado no projeto.

5. Conclusões e Perspectivas Futuras

A reutilização pode trazer diversos benefícios no desenvolvimento de software, especialmente se praticada em níveis mais altos de abstração, como no processo de Engenharia de Requisitos (ER). Contudo, diversos fatores como, por exemplo, a falta de definição de um processo visando à reutilização e de ferramentas que apóiem tal processo, têm inibido uma reutilização mais efetiva. Visando a contribuir para a institucionalização de práticas de reúso na ER, este trabalho apresentou um processo de ER baseado na reutilização de ontologias e padrões de análise.

De sua aplicação inicial, conforme relatos dos envolvidos, pode-se verificar benefícios do reúso, com destaque para a elaboração de modelos conceituais apoiados, sobretudo, por conceituações explicitadas em ontologias.

Dentre os pontos fracos detectados estão a falta de um apoio mais efetivo ao reúso na modelagem de casos de uso e a dependência de uma seleção prévia de modelos para reúso. Em relação ao primeiro ponto, espera-se avançar na pesquisa para estabelecer meios de se utilizar ontologias de tarefas na derivação de casos de uso e suas descrições. No segundo caso, estamos cientes de que esse é um dos grandes problemas da reutilização como um todo: a seleção de itens a serem reutilizados. Acreditamos que por meio de técnicas e ferramentas de gerência de conhecimento é possível dar um passo à frente para a solução deste problema e estamos trabalhando em um ambiente integrado de apoio à ER capaz de sugerir pró-ativamente ontologias e padrões de análise a serem reutilizados. Vale destacar que, ainda que nosso foco neste artigo esteja no desenvolvimento *com* reúso, sabemos que ele só é possível com um desenvolvimento *para* reúso. Assim, um ambiente de apoio à ER deve considerar as duas perspectivas.

6. Agradecimentos

Este trabalho foi realizado com o apoio do CNPq e da CAPES, entidades do Governo Brasileiro dedicadas ao desenvolvimento científico e tecnológico, e da FAPES, Fundação de Apoio à Ciência e Tecnologia do Espírito Santo. Agradecemos, ainda, à Acropolis Soluções em Tecnologia da Informação por participar do estudo e permitir sua divulgação.

7. Referências Bibliográficas

[1] Gimenes, I.M.S., Huzita, E.H.M. (2005) *Desenvolvimento Baseado em Componentes: Conceitos e Técnicas*. Editora Ciência Moderna.

[2] Hofmann, H.F., Lehner, F. (2001) "Requirements Engineering as a Success Factor in Software Projects", IEEE Software, July/August.

RF001 - O sistema deve apoiar a alocação de colaboradores a projetos de desenvolvimento e manutenção de acordo com a equipe em que cada colaborador se encontra.				
Dependências	Conflitos	Caso de Uso	Classes	Origem da Classe
RF004 RF005 RF008		Alocar Colaborador	`Atividade`	Conceito `Atividade` - Ontologia de Processo [19]
			`Alocacao`	Classe Associativa `Alocacao` - Padrão "Alocação de Recurso" [16]
			`Pessoa`	Conceito `Pessoa` - Ontologia de Organização [18]
			`Projeto`	Conceito `Projeto` - Ontologia de Organização [18]
			`Processo`	Conceito `Processo de Projeto` - Ontologia de Processo [19]
			`Equipe`	Conceito `Equipe` - Ontologia de Organização [18]

Figura 6 – Parte do Relatório de Rastreabilidade do Projeto

[3] Zave, P. (1997) Classification of research efforts in requirements engineering**.** ACM Computing Surveys Journal, vol. 29, n. 4, p. 315-321.

[4] Kotonya, G., Sommerville, I. (1998) *Requirements engineering: processes and techniques*. Chichester, England: John Wiley.

[5] Lamsweerde, A. V. (2000) Requirements engineering in the year 00: a research perspective. Proceedings of the 22nd International Conference on Software Engineering - ICSE´2000, Limerick, Ireland. p. 5-19.

[6] Werner, C.M.L., Braga, R.M.M. (2005) A Engenharia de Domínio e o Desenvolvimento Baseado em Componentes. In: Gimenes, I.M.S., Huzita, E.H.M. *Desenvolvimento Baseado em Componentes: Conceitos e Técnicas*. Editora Ciência Moderna.

[7] Fowler, M. (1997) *Analysis Patterns: Reusable Object Models***.** Addison-Wesley Professional Computing Series.

[8] Robertson, S., Robertson, J. (1999), *Mastering the Requirements Process*, Addison Wesley, ACM Press.

[9] Devedzic, V. (1999) "Ontologies: Borrowing from Software Patterns"**.** Intelligence, vol. 10, issue 3 (Fall 1999), p. 14-24.

[10] Falbo, R.A., Guizzardi, G., Duarte, K.C. (2002) "An Ontological Approach to Domain Engineering". Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering, SEKE'2002, Ischia, Italy, p. 351- 358.

[11] Noy, N.F., Hafner, C.D. (1997) The State of Art in Ontology Design: A Survey and Comparative Review. AI Magazine.

[12] Cima, A. M., Werner, C. M. L. (1997) A Reutilização de Conhecimento Abstrato e a Análise de Domínio**.** Relatório Técnico ES-432/97 – Engenharia da Sistemas e Computação, COPPE, Universidade Federal do Rio de Janeiro.

[13] Guizzardi, G. (2000) Desenvolvimento para e com Reuso: Um Estudo de Caso no Domínio de Vídeo sob Demanda. Dissertação (Mestrado em Informática), Universidade Federal do Espírito Santo (UFES).

[14] Nuseibeh, B., Easterbrook, S. (2000), "Requirements Engineering: A Roadmap", In: Proc. of the Future of Software Engineering, ICSE'2000, Ireland, p. 37-46.

[15] Gruber, T.R. (1995) Toward principles for the design of ontologies used for knowledge sharing. *Int. Journal Human-Computer Studies*, 43(5/6), p. 907-928.

[16] Cota, R.I. (2003) Um Estudo sobre o Uso de Ontologias e Padrões de Análise na Modelagem de Sistemas de Gestão Empresarial. Dissertação de Mestrado, Mestrado em Informática, UFES.

[17] Lima, K.V.C. (2004) Definição e Construção de Ambientes de Desenvolvimento de Software Orientados a Organização. Tese de Doutorado, COPPE/UFRJ.

[18] Ruy, F.B. (2006) Semântica em um Ambiente de Desenvolvimento de Software. Dissertação de Mestrado, Mestrado em Informática, UFES.

[19] Bertollo, G. (2006) Definição de Processos em um Ambiente de Desenvolvimento de Software. Dissertação de Mestrado, Mestrado em Informática, UFES.

Elicitación de Requisitos empleando UN-Lencep y Esquemas Preconceptuales[1]

Carlos Mario Zapata J.
Grupo de Ingeniería de Software, Escuela de Sistemas, Universidad Nacional de Colombia
cmzapata@unal.edu.co

Fernando Arango I.
Grupo de Ingeniería de Software, Escuela de Sistemas, Universidad Nacional de Colombia
farango@unal.edu.co

Resumen

Con la Elicitación de Requisitos se pretende la captura de las necesidades de los interesados y su traducción en especificaciones del software por construir. Las técnicas tradicionales de elicitación de requisitos tienen una alta incidencia del analista y son difíciles de automatizar, debido a la subjetividad del proceso. Por ello, desde mediados de los noventa se viene presentando una tendencia hacia la automatización del proceso de elicitación de requisitos, especialmente para especificaciones en diagramas de UML; en estos trabajos aún subsisten problemas, pues no se obtienen todos los tipos de diagramas de UML y se presentan problemas de consistencia entre los diagramas resultantes. En este artículo se presenta una técnica de elicitación basada en el lenguaje controlado UN-Lencep y los denominados Esquemas Preconceptuales, como una forma de solución a los problemas remanentes. Esta técnica se ejemplifica con un caso de estudio.

Abstract

Requirements Elicitation is intended for capturing stakeholder's needs and for translating them into software specifications. Traditional Requirements Elicitation Techniques have high analyst incidence and they are difficult to automate. From the mid-nineties there is a trend in requirements elicitation in an automated way, especially for UML diagrams; there are still problems with these works: many types of UML diagrams are not obtained, and consistency problems for resulting diagrams are raised. In this paper we present an elicitation technique based on UN-Lencep and the so-called Pre-conceptual Schemas. This technique tries to solve remaining problems, and it is also exemplified with a case study.

1. Introducción

El término Elicitación de Requisitos proviene presumiblemente de Leite [1] y se suele referir al proceso inicial de la Ingeniería de Requisitos. En este proceso se identifican, depuran e integran las necesidades de los interesados y, mediante un proceso de análisis, se traducen en especificaciones que usan lenguajes formales y semiformales. Uno de los lenguajes semiformales que se suele utilizar para dichas especificaciones es el UML [2].

La Elicitación de Requisitos ha empleado tradicionalmente una serie de técnicas que varían desde entrevistas hasta elaboración de prototipos [3–7], las cuales se suelen realizar conjuntamente entre los interesados en el desarrollo de una pieza de software (así llamados por el vocablo inglés *stakeholder*) y los analistas. En estas técnicas, la subjetividad y experiencia del analista son fundamentales para el éxito del proceso, lo cual genera inconvenientes si se busca una automatización del mismo. Por ello, desde mediados de la década pasada existe una nueva tendencia en Elicitación de Requisitos que procura su automatización total o parcial; en esta tendencia se procura la captura de las necesidades del interesado en lenguajes controlados, con el fin de obtener automáticamente los diagramas que especifican la pieza de software. Sin embargo, en esta tendencia aún cuenta con algunos problemas: en algunos casos el proceso aún tiene alta incidencia del analista [8] y en otros casos se obtiene un solo diagrama [9, 10] o se obtienen varios diagramas pero a través de procesos independientes que pueden generar problemas de consistencia entre los diagramas resultantes [11].

En este artículo se presenta una técnica para la Elicitación de Requisitos basada en un lenguaje controlado llamado UN-Lencep [12] y una representación intermedia del conocimiento

[1] Este artículo se realizó en el marco de los siguientes proyectos de investigación: "CONSTRUCCIÓN AUTOMATICA DE ESQUEMAS CONCEPTUALES A PARTIR DE LENGUAJE NATURAL", financiado por la DIME y "DEFINICIÓN DE UN ESQUEMA PRECONCEPTUAL PARA LA OBTENCIÓN AUTOMÁTICA DE ESQUEMAS CONCEPTUALES DE UML" y "CONSTRUCCIÓN DE UN EDITOR GENÉRICO PARA ELABORACIÓN DE MODELOS GRÁFICO SIMBÓLICOS, ETAPA III, Generación de un mecanismo para la Transformación entre Modelos", financiados por DINAIN y administrados por la DIME

denominada Esquemas Preconceptuales [13]. Con esta técnica se obtienen tres tipos de diagramas de UML.

Este artículo está organizado así: en la Sección 2 se discuten las técnicas tradicionales y modernas para realizar el proceso de Elicitación de Requisitos, en la Sección 3 se discuten los fundamentos teóricos de la propuesta, en la Sección 4 se presenta la técnica basada en UN-Lencep y Esquemas Preconceptuales para la Elicitación de Requistos, en la Sección 5 se presenta un caso de estudio para la aplicación de esta técnica y en la Sección 6 se discuten las conclusiones y trabajos futuros.

2. Técnicas para la Elicitación de Requisitos

2.1. Técnicas tradicionales

Para la Elicitación de Requisitos se han utilizado varias técnicas [3–7] entre las que se destacan las entrevistas y tormentas de ideas como las de más común uso. También se han usado variaciones de las entrevistas como el despliegue de la función de calidad (DFC), que también incluye revisión de documentación física de la organización y la técnica para facilitar la especificación de aplicaciones (TFEA), que se suele hacer para definir los requisitos de la aplicación. En otras técnicas el equipo de desarrollo asume provisionalmente el papel de interesado para comprender el dominio (como en el caso del juego de roles) o para definir una posible solución (como en el caso de la introspección). Un último grupo de técnicas tradicionales utiliza ciertos recursos especializados para definir conjuntamente con el interesado las características de la solución; entre estas técnicas se destacan los casos de uso o escenarios, el desarrollo conjunto de aplicaciones (Joint Application Development—JAD), el tablero de historias (StoryBoarding—SB) y el prototipado.

Las principales dificultades asociadas con estas técnicas tradicionales son:

- Todas las técnicas incluyen una alta participación del analista, lo que en general dificulta su automatización.
- Algunas técnicas como DFC y JAD incluyen la revisión de la documentación de la organización, que en general requeriría herramientas robustas de procesamiento de lenguaje natural en caso de ser automatizadas.
- Otras técnicas como TFEA, casos de uso, introspección, SB y prototipado tratan de establecer una versión preliminar de la solución, que es el objetivo de la Elicitación de Requisitos, pero no el punto de partida para el proceso de Elicitación.
- Finalmente, técnicas como casos de uso y JAD exigen del interesado conocimientos técnicos que en general no suelen poseer.

2.2. Métodos semiautomáticos y automáticos para la construcción de diagramas de UML

2.2.1. LInguistic Domain Analysis (LIDA). Es una herramienta semiautomática que permite la elaboración de diagramas de clases [8]. LIDA realiza una clasificación de las palabras incluidas en un discurso en lenguaje natural sobre el dominio, en verbos, sustantivos y adjetivos, con sus respectivas frecuencias de aparición; el analista utiliza esta información para elaborar el diagrama de clases asistido por la herramienta y finalmente puede complementar el diagrama con su propia experiencia. En este caso se presenta una alta incidencia del analista en la elaboración del diagrama.

2.2.2. Natural Language-Object-Oriented Product System (NL-OOPS) y Class Model Builder (CM-Builder). Estas dos herramientas permiten la obtención automática de diagramas de clases a partir de lenguajes controlados. Estos proyectos emplean redes semánticas como representaciones intermedias entre lenguaje natural y los esquemas conceptuales [9] y [10]; algunas de sus desventajas son:

• Sólo obtienen el diagrama de clases (y no otros diagramas UML).

• La representación intermedia mediante redes semánticas no permite representar las características dinámicas del modelo del discurso.

2.2.3. Natürlichsprachliche InformationsBedarfs-Analyse (NIBA), Análisis de Requisitos de Información en Lenguaje Natural. Este es quizá el proyecto más completo en esta tendencia. Este proyecto permite la obtención de diferentes diagramas UML (especialmente clases y actividades, aunque establecen que podrían obtener otros como secuencias y comunicación) a partir de un lenguaje controlado. Para ello, emplea un conjunto de esquemas intermedios que se denomina KCPM (Klagenfurt Conceptual Predesign Model), el cual posee formas diferentes de representación del conocimiento para los diferentes diagramas de UML, variando desde tablas con información especial para el diagrama de clases, hasta unos diagramas dinámicos especiales para el diagrama de actividades [11]; esto puede ocasionar ciertas pérdidas de información entre diagramas y, consecuentemente, fallas de consistencia entre los

mismos. Así pues, la información de tipo estático y dinámico no se puede mezclar para obtener una representación única que se pueda mapear a los diferentes diagramas UML.

3. Fundamentos Teóricos de la Solución

En la Sección 2.2. se discutieron algunos métodos para la construcción—con algún nivel de automatización—de los diagramas de UML a partir de un lenguaje natural o controlado. En los métodos completamente automáticos el uso de lenguajes controlados facilita la interpretación de los textos, pues sólo se permiten estructuras muy sencillas (por ejemplo sujeto-verbo-complemento) que estén libres de ambigüedades. Para efectos de esta propuesta, se diseñó UN-Lencep (Universidad Nacional de Colombia—Lenguaje Controlado para la Especificación de Esquemas Preconceptuales) [12], un lenguaje que admite un conjunto reducido de "plantillas" de frases para describir el dominio de un interesado. El objetivo principal de UN-Lencep es generar un medio de comunicación entendible por el interesado, pero que pueda ser validado por el analista, con el fin de que su trabajo conjunto repercuta en la obtención de mejores diagramas de UML de manera automática. La especificación de UN-Lencep se discute en la Sección 4.2.

Ahora, las propuestas completamente automáticas—presentadas en las Secciones 2.2.2. y 2.2.3.—emplean representaciones intermedias para realizar la transición del lenguaje controlado al UML. En esos trabajos, la representación intermedia es completamente dependiente del tipo de diagrama que se quiera obtener: se trata de una representación estructural cuando se dirige al diagrama de clases, y dinámica cuando se trata de diagramas de interacción. Para diagramas de comportamiento no se tienen representaciones intermedias en estos trabajos. El hecho de manejar representaciones independientes para los diferentes tipos de diagramas puede presentar problemas de consistencia al momento de la generación automática de esos diagramas. Por ello, en esta propuesta se optó por definir una representación unificada para los tres tipos de diagramas, que contuviera lo expresado en UN-Lencep pero que, a la vez, expresara las características de los diferentes diagramas. Esa representación se realiza mediante los denominados Esquemas Preconceptuales (EPs) [13], cuya especificación se presenta en la Sección 4.1.

En relación con el UML generado, se seleccionaron tres diagramas objetivo: uno estructural (diagrama de clases), uno comportamental (diagrama de máquina de estados) y uno de interacción (diagrama de comunicación). La elección de estos diagramas obedeció a su frecuencia de uso en las labores de desarrollo de distintas piezas de software; además, la implementación del software requiere que se conozcan las características de los tres tipos de diagrama.

4. UN-Lencep y Esquemas Preconceptuales para la Elicitación de Requisitos

4.1. Simbología de los Esquemas Preconceptuales

Los símbolos que emplean los EPs se muestran en la Figura 1 (tomada de [13] y complementada).

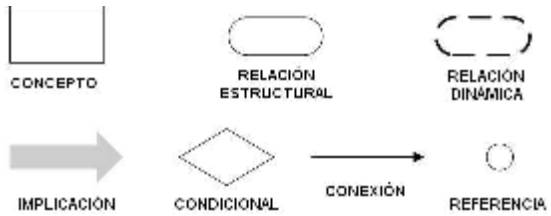


Figura 1. Símbolos de los Esquemas Preconceptuales.

La semántica asociada con estos símbolos es la siguiente:

• Conceptos: Son sustantivos del discurso del interesado; también pueden ser sintagmas nominales del tipo sustantivo-preposición-sustantivo, como por ejemplo "departamento de pedidos". Cada concepto aparece sólo una vez en cada EP, por lo cual, a medida que se generan representaciones de un concepto, se van sumando relaciones a ese concepto.

• Relaciones estructurales: Son verbos que generan conexiones permanentes entre los conceptos. En esta categoría sólo se reconocen los verbos "es" y "tiene".

• Relaciones dinámicas: Son verbos que denotan acciones u operaciones en el mundo. Algunos ejemplos son: "registra", "paga", "presenta", etc.

• Implicaciones: Expresan relaciones de causa-efecto entre relaciones dinámicas o entre condicionales y relaciones dinámicas. Tienen el mismo significado de la implicación lógica, en la cual se requiere que el antecedente se cumpla (ya sea relación dinámica o condicional) para que el consecuente (siempre una relación dinámica) se cumpla.

• Condicionales: Son expresiones conformadas por conceptos y operadores entre ellos que sirven como precondición a una relación dinámica.

• Conexiones: Son flechas que unen los conceptos con relaciones dinámicas o estructurales y viceversa.

• Referencias: Son círculos numerados que permiten ligar elementos físicamente distantes en el diagrama.

4.2. Especificación de UN-Lencep

En la Tabla 1 se muestra la construcción formal del UN-Lencep [12] con algunas expresiones de equivalencia en lenguaje natural controlado; en la Figura 2 se muestran las reglas de conversión de la especificación básica de UN-Lencep en EPs.

Tabla 1. Construcción Formal del UN-Lencep.

Construcción Formal	***Expresión en Lenguaje Natural Controlado***	
A <ES> B	A *es una especie de* B A *es un tipo de* B	A *es una clase de* B B *se divide en* A
A <TIENE> B	A *incluye* B A *contiene* B A *posee* B A *está compuesto por* B A *está formado por* B B *pertenece a* A	B *es una parte de* A B *está incluido en* A B *está contenido en* A B *es un elemento de* A B *es un subconjunto de* A
A <R1> B	<R1> puede ser cualquier verbo dinámico, por ejemplo: A *registra* B, A *paga* B	
C <R2> D, <SI> A <R1> B	*si* A <R1> B *entonces* C <R2> D *Dado que* A <R1> B, C <R2> D *Luego de que* A <R1> B, C <R2> D	
<SI> {COND} <ENTONCES> A <R1> B, <SINO> C <R2> D	{COND} es una condición expresada en términos de conceptos. <R1> y <R2> son verbos dinámicos. <SINO> es opcional, por ejemplo: *si* M *es mayor que 100 entonces* A *registra* B	

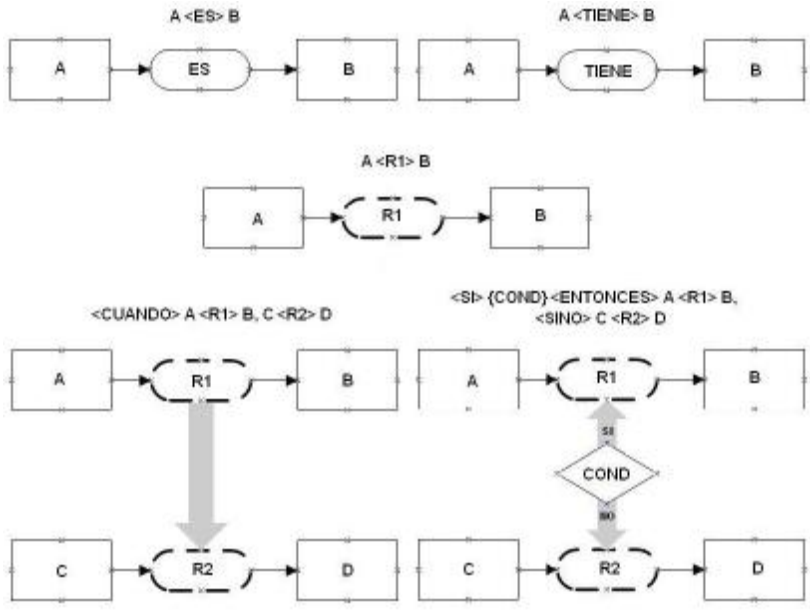


Figura 2. Reglas de conversión de la especificación básica de UN-Lencep en EPs

A partir de las reglas incluidas en la Tabla 1 y la Figura 2 es posible realizar la descripción de un dominio cualquiera mediante expresiones en Lenguaje Natural Controlado. Posteriormente, estas expresiones se pueden traducir a la construcción formal de UN-Lencep, para luego ser traducidas a EPs.

4.3. Reglas para obtención de UML a partir de EPs

El inventario de los elementos de los diagramas UML que se identifican a partir de los EPs se compendia en la Figura 3.

Figura 3. Inventario de elementos de UML 2.0 que se identifican a partir de un discurso en EPs.

Adicionalmente, la expresión gráfica de algunas de las reglas que permiten dicha obtención se compendian en la Tabla 2. Las reglas completas se pueden consultar en [13]. En la columna "Precondición" se encuentran combinados elementos de los Esquemas Preconceptuales con elementos de otros diagramas, puesto que algunas reglas emplean de manera recursiva los elementos que se van identificando a lo largo del proceso (véanse, por ejemplo, las reglas 3 y 5).

5. Caso de Estudio

Con base en los fundamentos teóricos de la Sección 3 y las especificaciones de la Sección 4, el Grupo de Ingeniería de Software de la Escuela de Sistemas de la Universidad Nacional de Colombia ha desarrollado la herramienta CASE UNC-Diagramador. Para su construcción se han empleado las ventajas de la tecnología .NET de Microsoft® y su lenguaje C#, combinándolas con las posibilidades gráficas de Microsoft Visio®. Además, el módulo de manejo del UN-Lencep se desarrolló en lenguaje PHP.

UNC-Diagramador realiza las siguientes funciones:

• Permite el ingreso de frases en lenguaje controlado que se convierten en un archivo en UN-Lencep.
• Convierte el archivo en UN-Lencep en el Esquema Preconceptual correspondiente.
• Obtiene automáticamente los diagramas de clases, comunicación y máquina de estados de UML 2.0, correspondientes al Esquema Preconceptual generado.

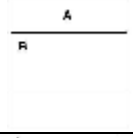
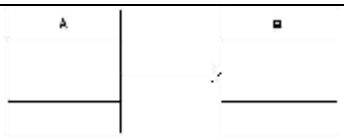
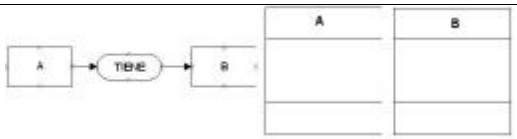
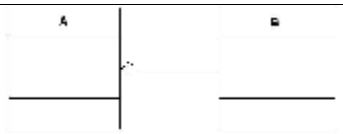
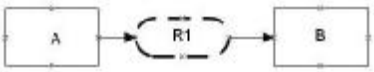
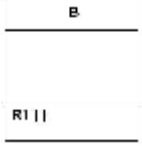
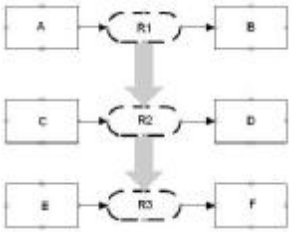
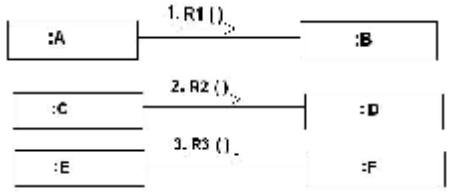
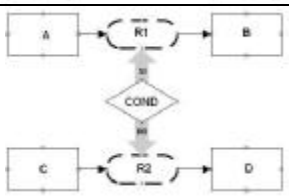
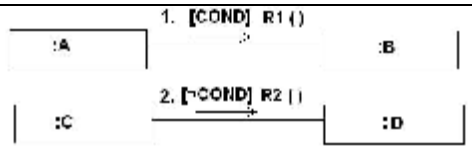
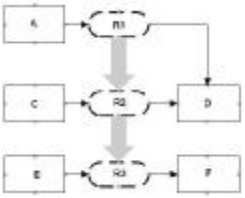
Seguidamente se presenta un conjunto de frases en lenguaje natural controlado que describen parcialmente el dominio de una clínica veterinaria. Este caso de estudio se usará para mostrar el funcionamiento del UNC-Diagramador para la generación automática de esquemas conceptuales de UML 2.0.

• El propietario posee una mascota
• La identificación es un elemento de una mascota
• Un nombre pertenece a una mascota
• Una mascota posee una historia_clínica
• El número es un elemento de una historia_clínica
• El detalle es un elemento de una historia_clínica
• La fecha es un elemento de detalle
• El detalle contiene un diagnóstico
• El detalle contiene un medicamento
• Dado que el propietario pide una cita, la secretaria asigna la cita
• Si el propietario cumple la cita, el veterinario revisa la mascota
• Luego de que el veterinario revisa la mascota, el veterinario registra el diagnóstico
• Si el veterinario registra el diagnóstico, entonces el veterinario receta un medicamento

Estas frases se deben introducir una a una en la interfaz que se muestra en la Figura 4, correspondiente al módulo de procesamiento de UN-Lencep.y se obtiene la especificación siguiente:

• ST propietario TIENE mascota
• ST mascota TIENE identificacion
• ST mascota TIENE nombre
• ST mascota TIENE historia_clinica
• ST historia_clinica TIENE numero
• ST historia_clinica TIENE detalle
• ST detalle TIENE fecha
• ST detalle TIENE diagnostico
• ST detalle TIENE medicamento
• IM Cuando PROPIETARIO PIDE CITA entonces SECRETARIA ASIGNA CITA
• IM Cuando PROPIETARIO CUMPLE CITA entonces VETERINARIO REVISA MASCOTA
• IM Cuando VETERINARIO REVISA MASCOTA entonces VETERINARIO REGISTRA DIAGNOSTICO
• IM Cuando VETERINARIO REGISTRA DIAGNOSTICO entonces VETERINARIO RECETA MEDICAMENTO

Tabla 2. Expresión Gráfica de las Reglas de obtención de los diagramas de UML a partir de los EPs.

No.	*Precondición*	*Resultado*
1		
2		
3		
4		
5		
6		
7		
8		

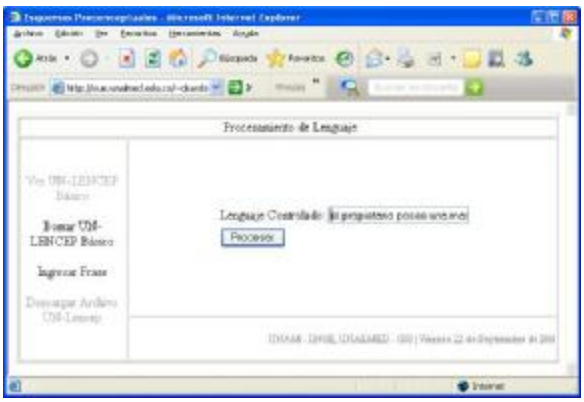


Figura 4. Interfaz Gráfica de Usuario del módulo de procesamiento de UN-Lencep.

La abreviatura ST denota una frase de tipo estructural e IM denota una frase de implicación. Esta especificación se puede leer con el UNC-Diagramador para obtener el Esquema Preconceptual que se muestra en la Figura 5. Finalmente, luego de presionar el ícono de UML ubicado en la parte superior derecha de la Figura 5, se obtienen los tres diagramas de UML 2.0 que se mencionan en las reglas de la Sección 4; estos diagramas se muestran en las Figuras 6, 7 y 8.

A nivel experimental, el UN-Lencep y los Esquemas Preconceptuales se han incorporado en el denominado UN-MÉTODO [14], el método de Elicitación de Requisitos que se enseña a los estudiantes de la Escuela de Sistemas de la Universidad Nacional de Colombia. Este método hace parte de la formación que se imparte en la Línea de Profundización en Ingeniería de Software, durante el primer curso denominado "Ingeniería de Requisitos". Durante el último año, se han elaborado 12 procesos de Ingeniería de Requisitos para diferentes desarrollos de software, que varían desde software institucional para la Universidad Nacional de Colombia hasta piezas de software de tipo comercial con aplicaciones en créditos y mantenimiento industrial, pasando por aplicaciones de carácter científico, como por ejemplo la medición de la confianza en grupos de personas y la automatización del diálogo analista-interesado en el proceso de captura de requisitos. En promedio, se utilizaron 30 conceptos para describir cada dominio y 40 relaciones entre estructurales y dinámicas. Pese a las limitaciones del UN-Lencep para la descripción de esos dominios, en todos los casos los interesados en el desarrollo de las piezas de software pudieron comprender y corregir los Esquemas Preconceptuales resultantes, así como aportar información adicional que no se había detectado en las entrevistas analista-interesado.

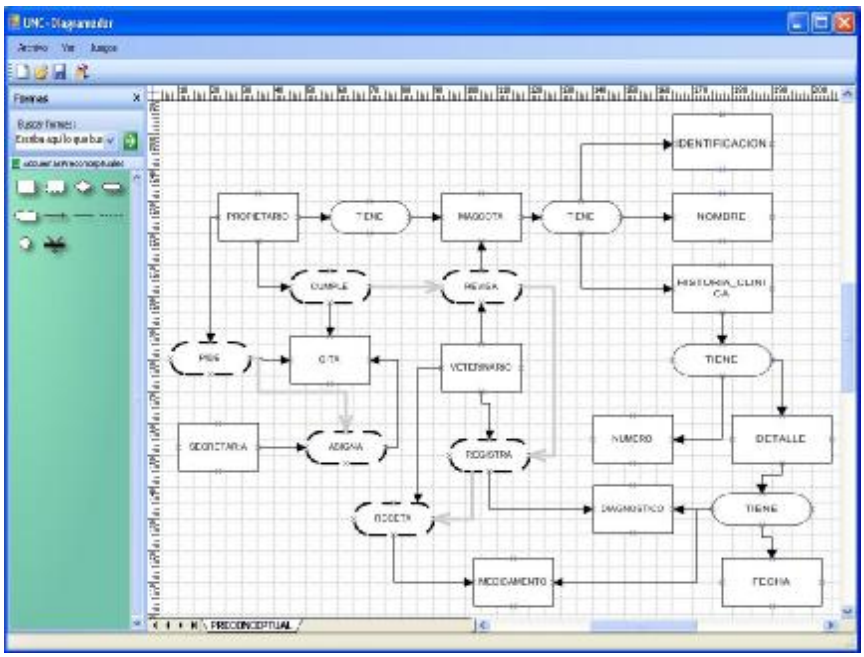


Figura 5. EP resultante de la especificación UN-Lencep introducida

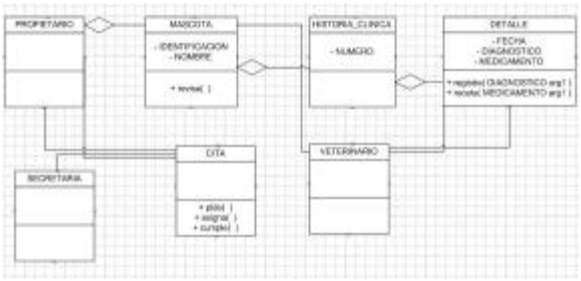


Figura 6. Diagrama de clases obtenido para el caso de estudio.

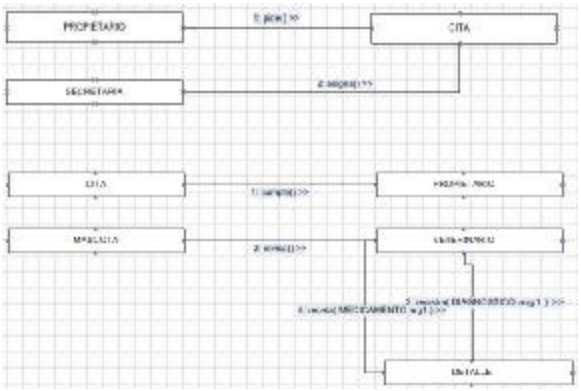


Figura 7. Diagramas de comunicación obtenidos para el caso de estudio.

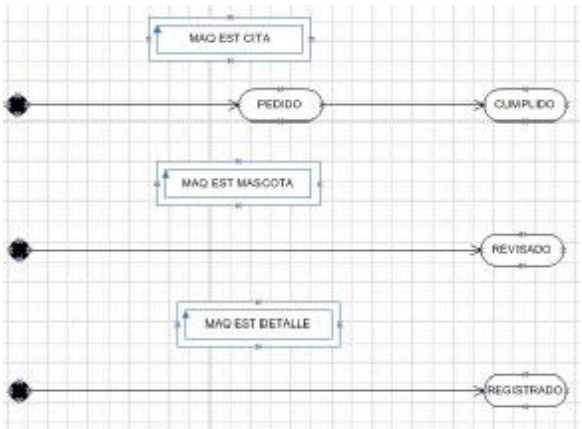


Figura 8. Diagramas de máquina de estados obtenidos para el caso de estudio.

Además, como resultado de la aplicación de las reglas de conversión que se ejemplifican en la Tabla 2 (y cuya implementación total hace parte de la herramienta UNC-Diagramador), ha sido posible la elaboración de conjuntos de diagramas para cada dominio que cumplen con las siguientes reglas de consistencia:

- Cada mensaje del diagrama de comunicación debe corresponder a una operación del diagrama de clases. El objeto destino del mensaje debe pertenecer a la clase que tiene la operación en el diagrama de clases.
- Cada objeto del diagrama de comunicación debe pertenecer a una clase del diagrama de clases.
- Cualquier condición de guarda del diagrama de máquina de estados debe pertenecer también al diagrama de comunicación.
- Cada actividad incluida en una transición del diagrama de máquina de estados debe corresponder a una operación del diagrama de clases.
- Cada máquina de estados debe pertenecer a una clase del diagrama de clases.

Estas reglas han sido algunas de las identificadas en [15] elementos que deben ser sujetos a pruebas para los diseños de piezas de software orientadas a objetos. Se identificaron estas reglas por ser los errores que se cometen comúnmente en el proceso de elaboración de los diagramas que se mencionan en este artículo (clases, comunicación y máquina de estados). Una de las ventajas del uso de UN-Lencep y Esquemas Preconceptuales para la Elicitación de Requisitos ha sido precisamente que los analistas e interesados no han tenido que manejar manualmente las reglas de consistencia mencionadas, porque éstas han sido aplicadas automáticamente a las descripciones en UN-Lencep.

Ahora, una segunda ventaja que se ha podido detectar en el uso de UN-Lencep y Esquemas Preconceptuales para la Elicitación de Requisitos, ha sido la posibilidad de comunicación entre los analistas y los interesados en un "mundo intermedio". Los interesados pueden tratar de describir el dominio de su aplicación en UN-Lencep y discutirlo con los analistas; éstos, a su vez, pueden tener ideas preconcebidas sobre el dominio de la aplicación que podrían estar representadas en su conocimiento técnico (por ejemplo, un analista por lo general puede saber si un elemento del mundo se podría traducir en una operación, un atributo, una condición, o cualquier otra primitiva de UML). Con el UN-Lencep y los Esquemas Preconceptuales, ambas visiones se pueden encontrar en esta representación intermedia y ser discutidas por analistas e interesados en este lenguaje común. Esto ha permitido solucionar los problemas típicos de validación de los diagramas de UML que se suelen presentar en los procesos de Elicitación de Requisitos, y que se suelen presentar por el desconocimiento por parte de los analistas de la sintaxis de los diagramas UML. En los trabajos realizados en la Línea de Profundización en Ingeniería de Software la validación

con los interesados ha sido posible mediante el uso de UN-Lencep y Esquemas Preconceptuales.

6. Conclusiones y Trabajo Futuro

La Elicitación de Requisitos es, en la actualidad, una labor en la cual los analistas siguen teniendo una alta incidencia y por ello existe una nueva tendencia hacia la automatización de diagramas, especialmente de UML.

Siguiendo esta tendencia, se ha presentado en este artículo una técnica de Elicitación de Requisitos que emplea el lenguaje UN-Lencep, los Esquemas Preconceptuales y un conjunto de reglas heurísticas, con el fin de obtener automáticamente tres diagramas específicos de UML 2.0 (Clases, Comunicación y Transición de Estados). Dichos diagramas se obtienen en una versión muy preliminar, lejos de las especificaciones detalladas del diseño, pero con sus principales primitivas conceptuales.

Esta técnica se incorporó en la herramienta CASE UNC-Diagramador, que actualmente está en desarrollo en la Escuela de Sistemas de la Universidad Nacional, sede Medellín. Igualmente, se ejemplificó la herramienta con un caso de estudio correspondiente a una clínica veterinaria.

La aplicación experimental del UN-Lencep y los Esquemas Preconceptuales hasta ahora se ha limitado a procesos de Elicitación de Requisitos empleando el denominado UN-MÉTODO, dentro de la Línea de Profundización en Ingeniería de Software de la Universidad Nacional de Colombia. Entre los resultados más destacados se incluyen el manejo automático de la consistencia entre los tres tipos de diagramas generados y la posibilidad de validación de los diagramas por parte del interesado, lo que no suele ser posible en los desarrollos normales de software debido a las dificultades de los interesados para comprender los diagramas de UML.

Las limitaciones del UN-Lencep para la representación de los diferentes dominios es, hasta ahora, una de las principales dificultades de este proceso. Por ello, existen algunas líneas de interés que pueden dar continuidad al presente trabajo, entre las que se cuentan:

• Definición de reglas heurísticas que permitan la identificación de nuevos elementos de los diagramas descritos, tales como las multiplicidades de las clases o las acciones "on entry" al interior de un estado del diagrama de máquina de estados.

• Definición de reglas heurísticas que posibiliten la conversión del Esquema Preconceptual a otros diagramas UML (secuencias, casos de uso, etc.).

• Enriquecimiento sintáctico de los EPs con nuevos elementos que permitan la representación de otros tipos de palabras (adjetivos, adverbios, etc.) que podrían incrementar la gama de los discursos posibles en UN-Lencep.

• Definición de reglas heurísticas que posibiliten la obtención automática de un discurso en UN-Lencep a partir de un documento en lenguaje natural.

Además, se hace necesaria la realización de un conjunto de pruebas experimentales que permitan verificar la validez del UN-Lencep y los Esquemas Preconceptuales para la representación de diferentes dominios, su facilidad de uso por parte de analistas e interesados y la corroboración de las pruebas de consistencia que toleran los diagramas resultantes.

Referencias

[1] J. C. Leite, "A survey on requirements analysis", Advanced Software Engineering Project Technical Report RTP-071, Department of Information and Computer Science, University of California at Irvine, 1987.

[2] OMG, UML 2.0 Specification, www.omg.org/UML

[3] Sommerville, I., *Software Engineering* (6th. Ed), Addison–Wesley Longman Publishing Co., Inc., Boston, 2001.

[4] Pressman, R., *Software Engineering: A Practitioner's Approach*, 6th ed. McGraw Hill, New York, 2005.

[5] A. Durán y B. Bernárdez, "Metodología para la Elicitación de Requisitos de Sistemas Software. Versión 2.3", Universidad de Sevilla. Informe Técnico LSI-2000-10. Departamento de Lenguajes y Sistemas Informáticos. Facultad de Informática y Estadística, Sevilla, 2005. http://www.lsi.us.es/~amador/publicaciones/metodologia_elicitacion_2_3.pdf.zip

[6] S. Raghavan, G. Zelesnik, y G. Ford, "Lecture Notes on Requirements Elicitation", Educational Materials, CMU/SEI-94-EM-10, Software Engineering Institute, Carnegie Mellon University, 1994.

[7] D. Leffingwell, y D. Widrig, *Managing Software Requirements: A Unified Approach*, Addison Wesley, Reading, 1999.

[8] S. P. Overmyer, B. Lavoie y O. Rambow, "Conceptual modeling through linguistic analysis using LIDA". En: Proceedings of ICSE, Toronto, Canada, Mayo 2001.

[9] L. Mich, "NL-OOPS: From Natural Natural Language to Object Oriented Requirements using the Natural Language Processing System LOLITA", Journal of Natural Language Engineering, Cambridge University Press, Vol. 2, No. 2, 1996, pp. 161–187.

[10] H. Harmain y R. Gaizauskas, "CM-Builder: An Automated NL-based CASE Tool". En: Proceedings of the fifteenth IEEE International Conference on Automated Software Engineering (ASE'00), Grenoble, France, 2000.

[11] NIBA Project, "Linguistically Based Requirements Engineering—The NIBA Project". En: Proceedings 4th Int. Conference NLDB'99 Applications of Natural Language to Information Systems, Klagenfurt, Austria, Junio de 1999, pp. 177–182.

[12] C. M. Zapata, A. Gelbukh, y F. Arango, "UN–Lencep: Obtención Automática de Diagramas UML a partir de un Lenguaje Controlado". En: Memorias del 3er Taller en Tecnologías del Lenguaje Humano del Encuentro Nacional de Computación, San Luis Potosí, Septiembre de 2006.
[13] C. M. Zapata, A. Gelbukh, y F. Arango, "Pre-conceptual Schema: a UML Isomorphism for Automatically Obtaining UML Conceptual Schemas", Research in Computing Science: Advances in Computer Science and Engineering, Vol. 19, 2006, pp. 3–13.
[14] F. Arango y C. M. Zapata, *UN-MÉTODO para la Elicitación de Requisitos de Software,* Carlos Mario Zapata (Ed.), Medellín, 2006.
[15] J. McGregor y D. Sykes, *A practical guide to testing object-oriented software,* Addison-Wesley, Boston, 2001.

Onto-DOM: A Question-Answering Ontology-Based Strategy for Heterogeneous Knowledge Sources

Mariel Alejandra Ale[1], Cristian Gerarduzzi[1], Omar Chiotti[1,2], Maria Rosa Galli[1,2]
[1]*CIDISI - UTN – FRSF, Lavaise 610 – Santa Fe - Argentina*
male@frsf.utn.edu.ar
[2]*INGAR-CONICET, Avellaneda 3657 - Santa Fe – Argentina*
{chiotti, mrgalli}@ceride.gov.ar

Abstract

Nowadays, there are a large number of Knowledge Management (KM) initiatives implemented in organizations, which often fail to manage the natural heterogeneity of organizational knowledge sources. Many approaches to KM have been only based on new information systems technologies to capture all the possible knowledge of an organization into databases that would make it easily accessible to all employees. To overcome heterogeneity, documentation overload and lack of context we propose Onto-DOM, a question-answering ontology-based strategy within a Distributed Organizational Memory. Onto-DOM is a portable question-answering system that accepts natural language queries and, using a domain ontology, transforms and contextualizes the query eliminating the inherent natural language ambiguity. At the same time, recovers those knowledge objects with the higher probability of containing the answer.

1. Introduction

There are already a large number of Knowledge Management (KM) initiatives implemented in organizations, which often fail to manage the natural heterogeneity of organizational knowledge sources. Instead, many approaches to KM have been only based on new information systems technologies to capture all the possible knowledge of an organization into databases that would make it easily accessible to all employees [19][23]. This philosophy of regarding knowledge as a "thing" that can be managed like other physical assets has not been quite successful for several reasons related to tacit knowledge capture and tacit-to-explicit knowledge conversion.

To overcome some of the most common problems in KM – heterogeneity of knowledge sources, documentation overload and lack of context - we propose Onto-DOM, a question-answering ontology-based strategy implemented within a Distributed Organizational Memory (DOM) that goes beyond information management providing a framework for semantic treatment of organizational knowledge sources.

Onto-DOM is a portable question-answering system that accepts natural language queries and, using a domain ontology, transforms and contextualizes the query eliminating the inherent natural language ambiguity. At the same time, recovers those knowledge objects with the higher probability of containing the answer. We say that Onto-DOM is portable because the time needed to implement it in a new domain is minimum, requiring just a change of the associate domain ontology. Onto-DOM combines, in a novel way, a series of techniques to "understand" the natural language query and map it to the semantic annotation done in the organizational knowledge sources.

Onto-DOM does an intensive use of domain ontologies in a number of key processes. The domain ontology is used as the core of the classification strategy of knowledge objects. This strategy selects ontological concepts as descriptors obtaining a homogeneous representation of objects structurally heterogeneous. The domain ontology is also used in query refinement, the reasoning process (a process of generalization/specialization using ontology classes and subclasses) and similarity resolution.

In section 2, we present related work regarding to Organizational Memories and knowledge sources annotation (more precisely documents). In section 3, we present our Onto-DOM architecture. In section 4, we discuss the knowledge representation strategy for semantic document treatment within Onto-DOM. In section 5, we present the question treatment strategy. Finally, in section 6; we present conclusions and future works.

2. Related Work

This section describes related work in two different research areas, namely organizational memories and document annotation.

2.1 Organizational Memories

A great deal of effort has gone into the creation of electronic media necessary to capture and store information and improve communications. Nevertheless, this is not enough for an effective KM implementation. Experience shows that few workers contribute to knowledge repositories (case bases, knowledge bases, etc.) or search knowledge in them, and in this way, knowledge generated through the normal execution of daily tasks is lost [26]. Three key factors can be mentioned as possible causes of not using knowledge repositories [22]. On the one hand, knowledge contribution to repositories requires an extra documentation effort for workers and unless they perceive an immediate benefit, the additional work is not justified. On the other hand, knowledge sharing requires a common mental frame between source and receiver, but people from different backgrounds have different knowledge structures and perspectives. Moreover, repositories design focus on contents and tends to provide little context of the knowledge they contain. Knowledge is, by definition, highly context dependant while every explicit representation generally causes context elimination [4]. Without contextual information, knowledge workers cannot fully understand or trust the knowledge source and therefore adopt it [2]. Finally, in most cases, it does not exist a culture that fosters knowledge exchange within organizations.

To face these drawbacks it is necessary to develop knowledge enabler information systems that provide a common framework to capture, increase, store, organize, analyze and share not only information and data but also knowledge [26]. Currently, Organizational Memories (OMs) are proposed as support for effectively using, handling and preserving knowledge over time and space – as much as possible - without human intervention [1]. From the organizational perspective, an OM can act as a tool for KM and gives support to three types of learning in organizations: individual learning, learning through direct communication, and learning using a knowledge repository [18]. An OM comprises a variety of knowledge sources where information elements of different kinds, structures, contents and media types are available [1] and should be able to control and access this heterogeneous knowledge sources according to the user's information needs.

Although the previous definition seems to suggest a centralized approach, the centralization of an OM presents some disadvantages related to the distributed nature of organizational knowledge and the high maintenance cost of a centralized structure. These reasons lead to consider a distributed organizational memory approach [30].

Additionally, in today organizations, many Knowledge Intensive Tasks (KITs), such as dealing with complexity, uncertainty and abstractions, must be performed. These tasks involve an effective combination of corporative competencies and a constant knowledge object availability. These organizations, therefore, have to efficiently manage their capabilities, create mechanisms to elicit innovation and collect ideas and other knowledge sources to cope with KITs [32]. Many authors have proposed explicit business process modeling as a means to represent context and facilitate the treatment of specific situation anticipating knowledge objects requirements [1][30]. However, knowledge intensive processes tend to be characterized for dynamic changes in their objectives, context and restrictions. This kind of processes often presents collaboration patterns and highly ad-hoc communications that make the detailed and previous planning of the KITs difficult and, at the same time, make this proactive approach unsuitable to give support to the dynamic information needs that are very common in KITs performance. For these cases, a reactive approach is necessary. In this paper we present a DOM with a reactive behavior, which let users ask for the needed information at any point of their daily activity.

We propose a strategy for semantic representation of knowledge sources (more precisely, documents) with a domain ontology overcoming this way two major problems already mentioned: documentation overload and lack of context.

2.2 Document Annotation

Most of the work in this area comes from the Semantic Web. One of the first attempts to allow semantic annotation of web documents was done with the SHOE system [29], enabling web page authors to manually annotate their documents with machine-readable metadata. Another tool for the insertion of semantic markups in a manual way was Ontobroker [9]. Nevertheless, manual annotation is an expensive process and often leads to a knowledge acquisition bottleneck [24]. To overcome this problem semi-automatic annotation of documents has been proposed.

AeroDAML [21] uses a pattern-based approach and is designed to map proper nouns and common relationships to corresponding classes and properties in DARPA Agent Markup Language (DAML) [13] ontologies. Armadillo [11] performs wrapper induction on web pages to mine web sites that have a highly regular structure using a pattern-based approach to find entities.

KIM platform [20] uses the SESAME RDF [6] for ontology and knowledge base storage and a modified version of the Lucene [13] keyword-based search engine. The information extraction component of semantic annotation is performed using components of the GATE toolkit [8]. SemTag [10] is the semantic annotation component of a comprehensive platform, called Seeker, for

performing large-scale annotation of web pages. It tags large numbers of pages with terms from a standard ontology. In KIM, as well as in SemTag, annotation is considered as the process of assigning to the entities in the text links to their semantic descriptions provided by the ontology, therefore the focus is on the recognition of named entities, categorization of the larger text fragments is out of the scope of these projects.

MnM [31] provides an environment to manually annotate a training corpus, and then feed the corpus into a wrapper induction system based on the Lazy-NLP algorithm. The resulting output is a library of induced rules that can be used to extract information from corpus texts. MUSE [25] was designed to perform named entity recognition and coreferencing. It is implemented using GATE framework [8]. Ont-O-Mat [15] is an implementation of the S-CREAM semantic annotation framework [17]. The information extraction component is based on Amilcare. Amilcare is machine learning-based and requires a training corpus of manually annotated documents.

Our approach defers from these tools in several aspects. It uses GATE and WordNet for NLP processing and a domain ontology to provide the necessary context for knowledge objects. Our strategy does not have a learning phase that has to be redone every time we move to a different domain; only a change of the domain ontology is needed. We believe that these characteristics make this strategy suitable for a Distributed Organizational Memory implementation where a large and variable numbers of domains are presented and where KIT´s knowledge needs are continuously changing.

3. Distributed Organizational Memories and Domain Ontologies for KM

Organizational knowledge is the collective sum of tacit and explicit knowledge within an organization and it can be found embedded in routines and processes that enable action. It is also knowledge captured by the organization's systems, processes, products, rules, and culture. These definitions are good conceptual notions about what organizational knowledge is, but they offer little guidance as how to acquire, manage and transfer it among entities within the organization [27]. Most of the times, KM is confused with Information Management. In Information Management, information is stored, usually in databases, sorted and retrieved. Knowledge, on the other hand, requires a system that not only can store the existing knowledge as information, but it also can retrieve and use that information as knowledge when needed. In this manner, new knowledge can be created from existing knowledge in combination with new information [16].

Some organizational KM systems proposals focus on the application of information technologies for the capture, storage, and retrieval of organizational knowledge. In these approaches, OMs are proposed as support for knowledge effective representation, use, handling and conservation through time and space - up to where it is possible - without human intervention.

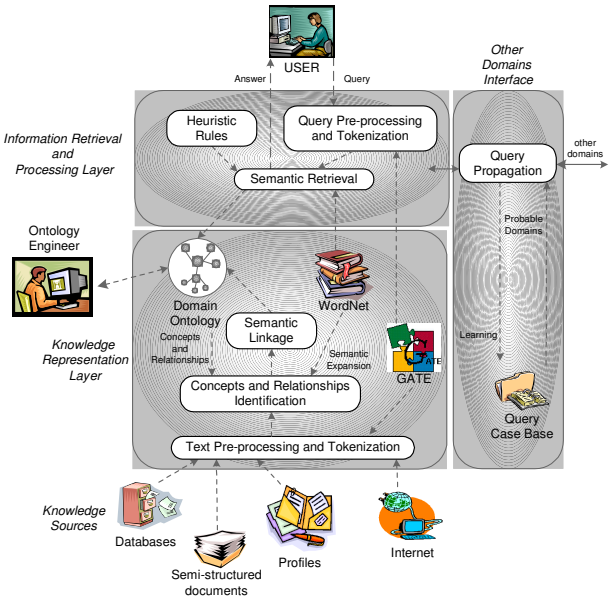


Fig. 1: Onto-DOM architecture

In this paper, OMs are defined as the means by which knowledge from the past is brought to bear on present activities resulting in higher level of organizational effectiveness [7]. As we said before, knowledge is distributed across the organization and it is necessary to represent and retrieve knowledge objects in the same manner. To this aim we propose to associate every organizational knowledge domain with its own OM and add an interface that enables knowledge retrieval from other domain OM if it is necessary [3]. In this particular type of OM, the characteristics, attributes, and semantics of the knowledge objects, as well as the relationships among them are represented through a domain ontology. Ontologies aim to capture domain knowledge in a generic way and provide a commonly agreed understanding of a domain, which may be reused, shared, and operationalized across applications and groups [12].

An additional benefit of ontology modeling is the context representation. Ontologies provide a domain model that allows knowledge objects to be seen in their context and this can be crucial for subsequent reinterpretation or use in a new task or project. As is shown in Figure 1, our Onto-DOM architecture has three main components:

- Information Retrieval and Processing Layer: it is responsible for user query analysis, query transformation to a matching format and information retrieval.
- Knowledge Representation Layer: this component is responsible for the knowledge extraction and representation from heterogeneous sources.
- Other Domains Interface: It is responsible for propagate the user query to other domains that can provide an answer. In order to accomplish this task the module implements a learning mechanism to propose possible target domains.

Another important advantage provided by ontologies can be seen in the Information Retrieval area, where the availability of an ontology allows replacing the traditional keyword-based retrieval approaches by more sophisticated ontology-based retrieval mechanisms [14][28]. In fact, ontologies are often presented as silver bullets for the Semantic Web [12] and are expected to bring several benefits to Information Retrieval related to recall and precision, user assistance in query formulation, and retrieval from heterogeneous knowledge sources.

In the next sections we will describe the implementation of the most important layers: knowledge representation and information retrieval.

4. Knowledge Representation Layer

As we said before, our goal is to represent in a homogenous way, knowledge sources that are heterogeneous in nature (more specifically documents).

We propose a strategy for semantic document representation where ontologies are used as the main structure for the classification process. Our proposal relies in the hypothesis that domain ontologies contain all the relevant concepts and relationships in a given domain although how ontologies are built up in the domain is out of the scope of this paper. As is shown in Figure 2 we have developed a strategy that comprises several steps.

```
SemAnS (document t)
text_preprocessing (t)
{ /* search nouns */
                N = identify_nouns (t)
                foreach n in N
                { aux = search_straight_instance (n,
Ont)
                  if aux = true
                    markup n with parent(n)
                  else
                  H = ask_hyper_wordnet (n)
                  foreach h in H
                  { aux1 = search_straight_instance (h,
O)
                    if aux1 = true
                      markup n with h
                    endif
                  }
                  endif
                }
linkage (t, Ont)
}
```

Fig. 2.: Knowledge Representation Strategy

To illustrate our strategy, we present an example using an extended version of the Travel[1] ontology that contains more than 120 concepts from the tourism area and an extract of a web page[2] of the same domain that is shown in Fig 3.

Fig. 3.: Example Document

4.1 Tokenization and Lexical-Morphological Analysis for Concepts Identification

This task is divided into two main phases: on the one hand, the tokenization of the text and, on the other hand, the lexical-morphological analysis of each token. The tokenization consists of dividing the text into single lexical tokens. This is an important task that involves activities such as sentence boundary detection, simple white space identification, proper name recognition, among others. After tokenization, a lexical-morphological analysis has to be done using a POS (Part-of-Speech) tool. In our case, we use the POS tagger provided by GATE[3] (General Architecture for Text Engineering) which specifies if a term is a verb, an adjective, an adverb, or a noun. The GATE platform has been widely used as a basis for Information Extraction processes and content annotation management. It provides the fundamental text analysis technologies on which we have constructed our strategy.

Usually, the decision on whether a particular word will be used as representative term is related to the syntactic nature of the word. In fact, noun words frequently carry more semantics than adjectives, adverbs, and verbs [5]. As, in our case, representative terms will be determined by ontological concepts, which are nouns, we will focus in this syntactic category within the tagged text.

[1] available at http://protege.stanford.edu/plugins/owl/owl-library/index.html (for the extended version send a request to male@frsf.utn.edu.ar)
[2] available at http://www.vacationidea.com/hotels/acqualina.html
[3] available at http://gate.ac.uk/

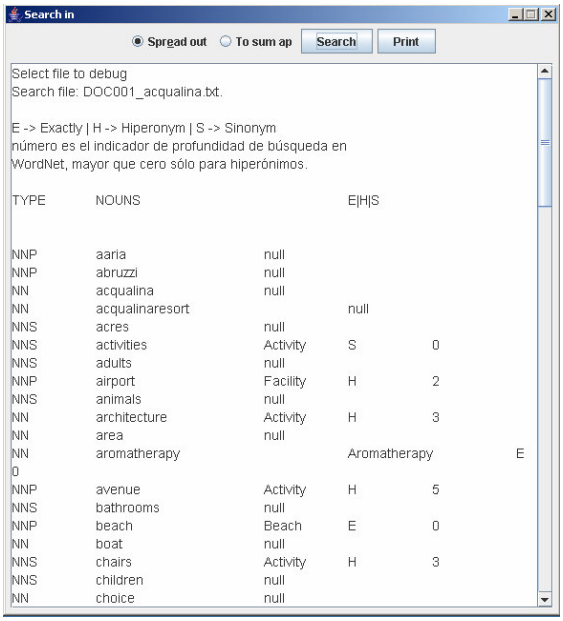


Fig. 4.: Knowledge Representation Prototype

In this sense, ontological concepts can be seen as possible classifying categories. At this stage, if the noun is not directly found in the ontology (search_straight_instance), using the synonyms set and hyperonymic/hyponymic structure provided by WordNet[4], we semantically expand every noun identified in the text and perform a search in the ontology (ask_hyper_wordnet). By doing this, we do not only identify exact ontological concepts occurrences but also derivations of the same word or even a synonym. Up to this point, we are not interesting in the meaning of each possible concept and that is why the presence of more than one sense for each noun in WordNet is not a problem.

For example, the concept "food" has been found with WordNet assistance. In this particular case, using WorldNet's hypernym relationship we found out that "meal" (a concept present in the text) is a kind of "food", which is a concept in the ontology. In other cases, this tool helps us to mark as ontological concept occurrences the presence of synonyms, and in this way, if the noun is not found directly in the ontology, WordNet allows us to expand the matching possibilities taking advantage of related concepts (synonyms, hypernyms, etc.) (Figure 4).

4.2 Semantic Document Representation

At this point, we navigate through the domain ontology using the properties structure in order to find relationships among previously identified concepts. By doing this, we aim to contextualize those concepts that, in another way, could not be related with other concept among the ones that were identified in the previous step.

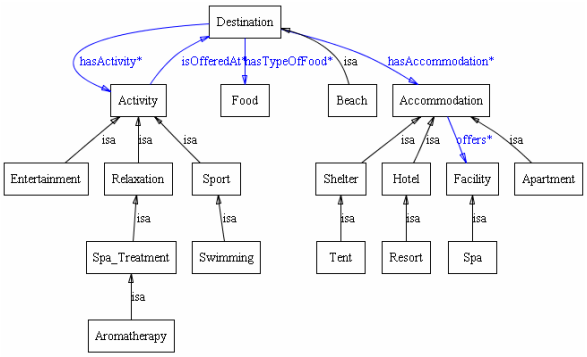


Fig. 5.: Ontology Representation

We take advantage of the ontological relationships in order to perform a more accurate and contextualize representation of the document and, as a result, we finally obtain the subset of the domain ontology that better models the document semantic content (Figure 5). This semantic document classification will enable new, semantically enhanced, access methods.

4.3 Representation Evaluation

As a first step in the implementation process, we have estimated the representation strategy performance applying the following metrics according to Yang´s [33] definitions: recall, precision, fallout, and accuracy[5].

Recall	Precision	Fallout	Accuracy
87%	70%	14%	86%

Recall is a measure of how well the strategy performs in finding relevant concepts. Recall is 100% when every relevant concept is annotated. In theory, it is easy to achieve good recall simply annotating every noun in the text. Therefore, recall for itself it is not a good measure of the quality of the strategy. Precision, on the other hand, is a measure of how well the strategy performs in not annotating not relevant nouns. Finally, fallout is the measure of how quickly precision drops as recall is increased, in other words, represents the portion of non-relevant concepts that were annotated. We have analyzed the reason of the relative low value of recall measure and found that 82% of the not annotated relevant concepts correspond to names of vacation destinations that were either, places not recognized by WordNet (i.e. Caicos) or types of destinations that were not taking into account in the domain ontology (i.e. islands, archipelago). We believe that recall can

[4] available at http://wordnet.princeton.edu/index.shtml

[5] Perform over 150 documents with 35.091 words

be improved using common vocabulary domain lists and enriching the domain ontology.

5. Information Retrieval and Processing Layer

Most of the work in ontology-based question-answering tends to focus in simple query expansion or in exploiting the availability of a knowledge base linked to the ontology to provide a precise answer. In the first case, we believe that this is a limited use of ontology potential and, in the second case, a vast knowledge base must be learnt in order to provide adequate answers. The effort required to feed all organizational knowledge in a knowledge base is prohibitive. Moreover, if precise answers are required this process cannot be fully automatized.

In our case, Onto-DOM accepts natural language queries and, using the domain ontology, transforms the query eliminating natural language ambiguity and recovering those knowledge objects with the higher probability of containing the answer. In a sense, this layer tries to find similarity between the query and the ontological concepts.

When similarity is evaluated in a taxonomy, the natural solution is to calculate the distance between the two concepts. In this way, the shorter the path is, the more similar the concepts. Nevertheless, this approach relies in the hypothesis that the links in the taxonomy represents uniforms distances and this is not always true.

Our strategy to determine similarity includes both conceptual and relationship similarity. The first step, is to transform the query in a format that facilitates ulterior evaluations and, to this aim, we apply part of the same strategy for document representation. After this stage, we have not only nouns that match ontological concepts but we also keep the verbs in order to evaluate relationship similarity and wh-words that give us an idea of the type of answer expected (time, location, person, etc.).

Essentially, we are trying to "understand" the question lying on the codified knowledge in the domain ontology, lexical resources as WordNet and GATE and heuristics associate to the treatments of wh-words.

Frequently, natural language queries carry certain structural and syntactic ambiguity that cannot be solved with general knowledge of the world as the knowledge contained in WordNet but, in a specific domain, only one of these interpretations may be true. The key here is to determine restrictions derived from the domain knowledge (modeled in the domain ontology) to apply them in the resolution of natural language ambiguities. When this ambiguity cannot be automatic solved the only reasonable course of action is to interact with the user to let him/her decide.

For example: in the query "Where can I eat Vegetarian dishes? after the first analysis we obtain the following useful information:

```
eat(Vegetarian, Food) (where, location)
```

In this case, the concept Food is derived from Dishes with the help of WordNet`s hyperonymy structure. Nevertheless, as we said before, our main objective is to go beyond a keyword search or the use of the domain ontology as a query expansion tool. To this aim, on the one hand, we will use the verbs detected in the query to look for semantic similarity related to relationships, and on the other hand, we will analyze the concepts related to those relationships to see if they are of the type expected according to the wh-word.

Following the previous example we recover the ontological concepts identified in the query along with their neighbors, Restaurant and Chef (Figure 6). To decide if one of this neighbors is useful to represent the query (and not search only by Food and Vegetarian) we evaluate the similarity between the verb in the query (eat) and the verbs in the relationships attach to the concepts identified (serve, specialize) using the synonym and correlate sets of WordNet.

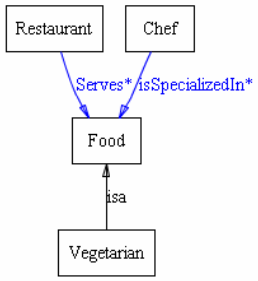


Fig. 6.: Ontological Concepts identified in the query (with their neighbors)

As it can be seen in Figure 7 "serve" has a higher semantic similarity with "eat" than "specialize".

To confirm this result, or as an alternative in the case that we are not able to obtain a conclusive result in the verbs comparison, we analyze the concepts in each end of the relationships (Restaurant, Chef) to see if they match with the type expected according to the wh-word. In this particular case, WordNet tells us that Restaurant is a Location (type expected) and Chef is a Person confirming that the portion of the domain ontology that better represents de query contains the concepts: Food, Vegetarian and Restaurant.

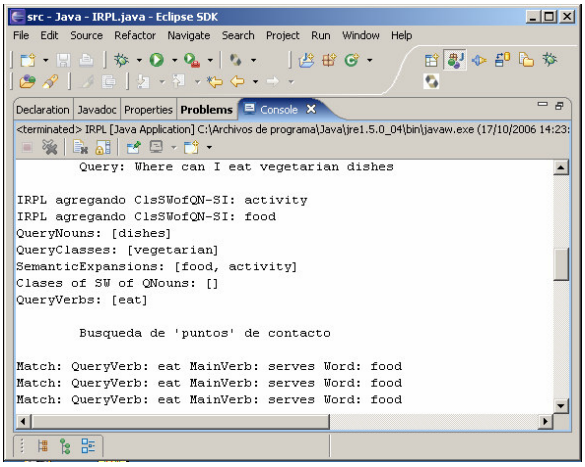


Fig. 7.: Relationship Similarity Analysis

6. Conclusions and Future Work

As we said before, our goal is to represent in a homogenous way, knowledge sources that are heterogeneous in nature (more specifically documents). To this aim, we have proposed a strategy for semantic document representation where ontologies are used as the main structure for the classification process. Our proposal relies in the hypothesis that domain ontologies contain all the relevant concepts and relationships in the domain.

Our initial experiments have yielded reasonable results. These show that it is possible to automatically perform operations such us document integration to an Organizational Memory by semantic annotation and a richer information retrieval. We provide here some observations gathered from the experiments but we defer a complete report to an extended version of this paper.

During our experiments with the annotation strategy we have identified several factors that may contribute to uncertainty. One of the reasons for errors in ontology concept identification has to do with the preprocessing of the text. This preprocessing includes a fully automatic noun markup that has an error rate that influence the effectiveness of the subsequent steps. These results could be improved using more sophisticated NLP techniques. The domain ontology definition, which is currently restricted to a relative small number of concepts, also contributes to a low recall rate.

Other problems identified are related to words association. For example, we found some documents that describe what things a place does not have (bars, theaters, cars, etc.) but the strategy classified these documents as describing places with those characteristics. We are currently seeking more advanced techniques to improve the analysis of negative expressions.

Finally, we have assumed the availability of a predefined domain ontology. This means that all documents will be treated based on that particular view of the world. However, in any realistic application scenario, new documents that have to be classified will generate the need for new concepts and relationships. Terms evolve in their meaning, or take on new meanings as organizational knowledge evolves. It is clear that we will have to find solutions to problems regarding with the addition, change or elimination of ontological concepts. A direction for further research will be the utilization of the annotation strategy to suggest ontology improvements.

Despite the issues to be solved as future work, our semantic representation strategy has proved to be a useful approach to solve two major problems in KM initiatives: documentation overload and lack of context. Our strategy is automatic and does not have a learning phase that has to be redone every time we move to a different domain; only a change of the domain ontology is needed. We believe that these characteristics make this strategy suitable for a DOM implementation where a large and variable numbers of domains are presented and where KIT´s knowledge needs are continuously changing. Finally, the query treatment strategy allows us to do a further use of ontology advantages that goes beyond query expansion.

7. References

[1]. Abecker, A.; Bernardi, A.; Hinkelmann, K.; Kühn, O. and Sintek, M.: Towards a Well-Founded Technology for Organizational Memories. *IEEE Intelligent Systems and their Applications*, Vol. 13, Issue 3, Page(s) 40 - 48, 1998.

[2]. Ackerman M. S.: Definitional and Contextual Issues in Organizational and Group Memories; *Proceedings of Twenty-seventh IEEE Hawaii International Conference of System Sciences* (HICSS 94), Page(s) 191-200; 1994.

[3]. Ale M., Chiotti O. and Galli M.R.: A Distributed Knowledge Management Conceptual Model for Knowledge Organizations; *ICFAI Journal of Knowledge Management*; ICFAI University Press; December 2005; Page(s) 27-39; 2005.

[4]. Apostolou D., Klein B., Traphöner R., Mentzas G., Jones S., Kyrloglou N. and Maass W.: INKASS Project: Intelligent knowledge asset sharing and trading, March 2002 – February 2004, http://www.inkass.com/. Last accessed October 2006.

[5]. Baeza-Yates R. and Ribeiro-Neto B.: Modern Information Retrieval; Addison-Wesley, Wokingham, UK, 1999.

[6]. Broekstra J., Kampman, A. and Harmelen F.: Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema in *International Semantic Web Conference* 2002, Sardinia, Italy, 2002.

[7]. Croasdell D., Jennex M., Yu Z. and Christianson T.: A Meta-Analysis of Methodologies for Research in Knowledge Management, Organizational Learning and Organizational Memory: Five Years at HICSS; *36th Annual Hawaii International Conference on System Sciences* (HICSS'03); Page 110; 2003.

[8]. Cunningham H., Maynard D., Bontcheva K. and Tablan V.:GATE: A Framework and Graphical Environment for Robust NLP Tools and Applications in *40th Anniversary Meeting of the Association for Computational Linguistics* (ACL'02), 2002.

[9]. Decker S., Erdmann M., Fensel D. and Studer R.: Ontobroker: Ontology based access to distributed and semi-structured information. In DS-8: *Database Semantics – Semantic Issues in Multimedia Systems*, IFIP TC2/WG2.6 Eighth Working

Conference on Database Semantics, Page(s) 351-369, Rotorua, New Zeland, 1999.
[10]. Dill S., Gibson N., Gruhl D., Guha R., Jhingran A., Kanungo T., Rajagopalan S., Tomkins A., Tomlin J.A. and Zien J.Y.: SemTag and Seeker: Bootstrapping the Semantic Web via automated semantic annotation in *Twelfth International World Wide Web Conference*, Budapest, Hungary; Page(s) 178-186; 2003.
[11]. Dingli A., Ciravegna F. and Wilks Y.: Automatic Semantic Annotation using Unsupervised Information Extraction and Integration in *K-CAP 2003 Workshop on Knowledge Markup and Semantic Annotation*, 2003.
[12]. Fensel D.: Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce. Springer Verlag; Berlín; 2001.
[13]. Greaves M.: DAML-DARPA Agent Markup Language; http://www.daml.org. Last accessed March 2006.
[14]. Guarino N., Masolo C. and Vetere G.: Ontoseek: Content-based access to the web; *IEEE Intelligent Systems*; Vol. 14, Issue 3; Page(s) 70-80; 1999.
[15]. Hackbarth G., and Grover V.: The knowledge repository: Organizational Memory information Systems; *Information Systems Management*, Vol. 16 Issue 3, Page(s) 21-30; (1999).
[16]. Hall D., Paradice D. and Courtney J.: Creating Feedback Loops to Support Organizational Learning and Knowledge Management in Inquiring Organizations; *Proceedings of the 34th Hawaii International Conference on System Sciences* (HICSS-34); Page(s) 10; 2001.
[17]. Handschuh S., Staab S. and Ciravogna F.: S-CREAM: Semi-automatic CREAtion of Metadata in SAAKM 2002 – *Semantic Authoring, Annotation & Knowledge Markup*, 2002.
[18]. Heijst G., Spek R. and Kruizinga E.: Corporate Memories as a tool for Knowledge Management; *Expert Systems with Applications*; Vol. 13, Issue 1; Page(s) 41-54; 1997.
[19]. King W.R.: Integrating Knowledge Management into IS strategy; I*nformation Systems Management*; Vol. 16 Issue 4; Page(s) 70-72; 1999.
[20]. Kiryakov A., Popov B., Terziev I., Manov D. and Ognyanoff D.: Semantic Annotation, Indexing and Retrieval; *Elsevier's Journal of Web Semantics*; Vol. 2 Issue 1; 2005.
[21]. Kogut P. and Holmes W.: AeroDAML: Applying Information Extraction to Generate DAML Annotation from Web Pages in *First International Conference on Knowledge Capture*, 2001.
[22]. Kwan M. and Balasubramanian P.: KnowledgeScope: managing knowledge in context; *Decision Support Systems*, Vol. 35, No. 4, Page(s) 467-486; 2003.
[23]. Levine L.: Integrating Knowledge and Processes in a Learning Organization; *Information System Management*; Page(s) 21-32; 2001.
[24]. Maedche A. and Staab, S.: Ontology Learning for the Semantic Web; *IEEE Intelligent Systems*; Vol. 16 Issue 2; Page(s) 72-79; 2001.
[25]. Maynard D.: Multi-source and Multilingual Information Extraction, *Expert Update*; 2003.
[26]. Nemati N., Steiger D., Iyer L. and Herschel R.: Knowledge warehouse: an architectural integration of knowledge management, decision support, artificial intelligence and data warehousing, *Decision Support Systems*, Vol. 33, Issue 2, Page(s) 143-161; 2002.
[27]. Nunamaker J., Romano N. and Briggs R.: A Framework for Collaboration and Knowledge Management; *Proceedings of the 34th Annual Hawaii International Conference on System Sciences* (HICSS 34); Vol. 1; Page 1060; 2001.
[28]. Richard-Benjamins V., Fensel D., Decker S. and Gómez-Pérez A.: (KA)2: building ontologies for the internet: a mid-term report; *International Journal of Human-Computer Studies*; Vol. 51, Issue 3, Page(s) 687-712; 1999.
[29]. Sean L., Lee S., Rager D. and Handler, J.: Ontology-based web agents. *Proceedings of the First International Conference on Autonomous Agents* (Agents'97); ed. Johnson, W.L. and Hayes-Roth, B.; Page(s) 59-68; USA; ACM Press; 1997
[30]. Sintek M., van Elst L., Lauer A., Maus H., Abecker A. and Schwarz S.: FRODO Project: A Framework for distributed organizational memories, January 2000 – December 2002, http://www.dfki.uni-kl.de/frodo/. Last accessed October 2006.
[31]. Vargas-Vera M., Motta E., Domingue J., Lanzoni M., Stutt A. and Ciravegna F.: MnM: Ontology Driven Semi-Automatic and Automatic Support for Semantic Markup in the *13th International Conference on Knowledge Engineering and Management* (EKAW 2002), Spain, Page(s) 379-391; 2002.
[32]. Vasconcelos J., Gouveia F. and Kimble C.: An organizational memory information system using ontologies; *Proceedings of the 3rd Conference of the Associação Portuguesa de Sistemas de Informação*; University of Coimbra, Portugal, 2002.
[33]. Yang, Y. : An evaluation of statistical approaches to text categorization. *Journal of Information Retrieval*; 1999, http://citeseer.ist.psu.edu/yang97evaluation.html. Last accessed October 2006.

Knowledge Engineering for a Fuzzy Power Plant Process Controller

Youngchul Bae[1], Yigon Kim[1], Malrey Lee[2], Sang Doo Shin[3] and Thomas Gatton[4]
[1]*Division Electronic Communication and Electrical Engineering*
Chonnam National University, Korea
[2] *The Research Center of Industrial TechnologyChonBuk National University, Korea*
mrlee@chonbuk.ac.kc
[3] *Korea East-West Power Co.,LTD Honam Thermal Power Plant, Korea*
[4] *School of Engineering and Technology, National University, La Jolla, CA 92037 USA*
tgatton@nu.edu

Abstract

This paper presents the knowledge engineering and development of a fuzzy controller using operating expertise and robust experimental numeric control data to replace the traditional PID controller and manual intervention in a thermal power plant main steam temperature control system. The temperature of the main steam temperature process must be uniformly controlled to maintain stable electric power output. Current process controller technology is plagued by hunting in cases involving various disturbances, resulting in switching to manual operation. To address this problem the Takagi-Sugeno-Kang (TSK) model is applied for fuzzy system control integrating fuzzy rules and information extracted directly from the real plant operating conditions. The fuzzy controller is implemented in a Distributed Control System (DSC) and its performance evaluated for feasibility using experimental simulation.

1. Introduction

Proper control of the superheated steam temperature process in thermal electrical power plants is very important in providing efficient operation and ensuring long equipment life. The superheated steam temperature control loop has a large time constant and overlapping interference from multiple variables. Especially, there are many related process variables introduced by the load variation and the operating pressure in the plant. In many of these cases, traditional PID control of the analog control system does not perform well and a microprocessor based digital distributed control system was applied to the power plant during the early 1980's. This control system has evolved from a simple and independent process to a more sophisticated control system and many optimization algorithms have been adopted for efficient process control. However, in many case, the variation of the operating points and system parameters, such as fuel types, sliding pressure operation, response characteristics of the actuators and the experience of the operators, trial and error methods have been used to fine tune the controller.

This paper presents a new fuzzy controller approach for the Distributed Control System (DCS) and its application to the existing digital control system. The fuzzy controller has characteristics of rapid and robust response, in comparison to traditional digital PID controller characteristics. The Takagi-Sugeno-Kang (TSK) model is adopted for the fuzzy controller [1] [2] and fuzzy clustering algorithms are used in the selection of the rules. A back-propagation neural network algorithm is used to determine the rule parameters to implement the fuzzy controller in the Distributed Control System (DCS) of the Master P-3000, LGIS power plant. The function block model for the controller is developed and tested to evaluate the performance of the proposed method. The test bed is composed of the DCS system and the simulator for the thermal power plant [3].

2. Steam temperature process fuzzy controller design

2.1. The superheated steam temperature control system

The purpose of the superheated (SH) control is to maintain a constant steam outlet temperature. Figure 1 illustrates the control system process. The process variable is the main steam outlet temperature of the super-heater and the value of the set point is a constant

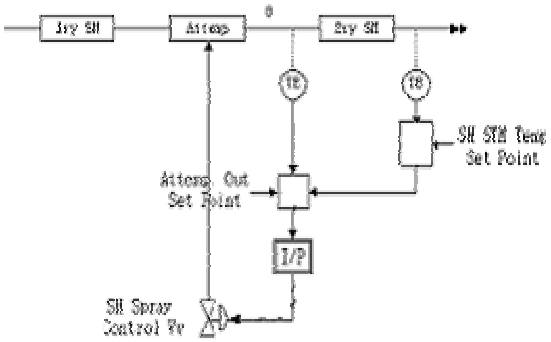


Figure1. The SH temperature control system

541˚C. Adjustable spray water valves are located in the inlet of the secondary super-heater and used to manipulate this control variable. The control logic is composed of the output of the main loop and cascaded to the secondary adjustable spray water valve control loop.

2.2. The fuzzy control system

The TSK model, adapted as the fuzzy controller model for the SH steam temperature control process, can be expressed as follows:

$$R^n: \text{If } x_1 \text{ is } o_1^n \text{ and } .... \text{ and } x_m \text{ is } o_m^n, \quad \text{Then } y^n = a_0^n + a_1^n x_1 + ... + a_m^n x_m \qquad (1)$$

where o_1^n represents a fuzzy set, n is the number of rules, m is the number of input variables, x is the input variable of the system, y is the output variable and a_m^n is the constant for the linear equation. The If part of the rule is equal to that of general if/then rules and the Then part is composed of the linear combination of the input variables. This fuzzy system approach is the most popular system and is suitable for real-time process control systems. The output of the control system can be calculated as follows:

$$F(x) = \frac{\sum_{i=1}^{M} y^i \mu^i}{\sum_{i=1}^{M} \mu^i} \qquad (2)$$

Where the compatibility value, $u^{l,}$ is represented by the equation:

$$\mu^i = \prod_{i=1}^{n} u_{c_i^l}(x_i) \qquad (3)$$

i

The membership function of the controller is a symmetric Gaussian formula and shown in Figure 2.

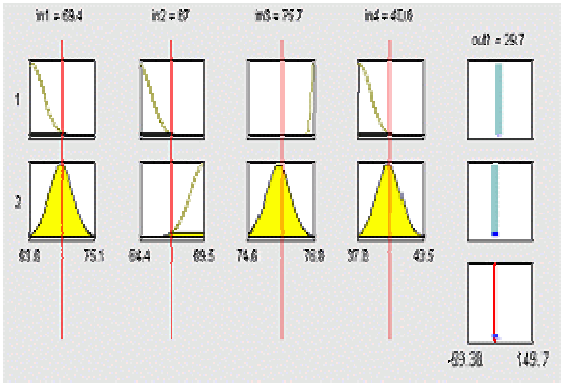


Figure2. Membership functions for the fuzzy inferential control system

The above inferential control system has two rules, four input variables and one output variable. Each membership function is characterized by the position of the center (c) and the standard deviation (s) of the following distribution function:

$$mf(x; s, c) = e^{-(x-c)^2/2s^2} \qquad (4)$$

The Gaussian membership function in Figure 3 shows the case when the center is 5 and the standard deviation is 2.

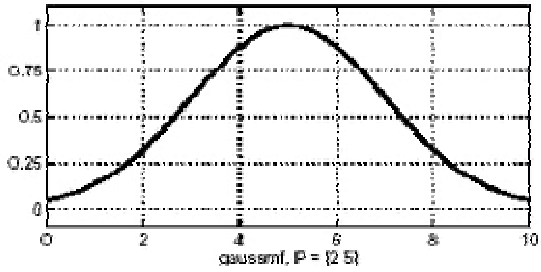


Figure 3. The symmetric Gaussian membership function

The fuzzy control system for the steam temperature control system using the fuzzy model, shown in Figure 4, is divided into two parts. The main part has 3 input variables, the main steam temperature (MST), the steam flow (SF) and the steam set point. The sub-part has 2 input variables, from the main output and the spray temperature, and 2 output variables for the spray valves (V.C.S). Blocks T1 and T2 represent the linear

transducer. Each control part is replaced by the fuzzy control system. The main part has 3 Gaussian fuzzy sets with rules and its sub-part has 4 Gaussian fuzzy sets and rules representing the data analysis.

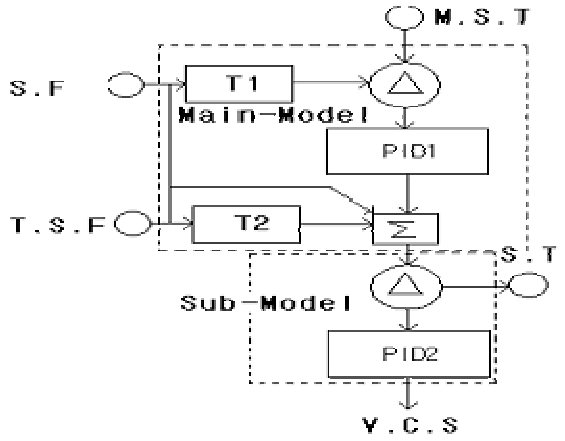


Figure 4. Structure for the main steam temperature control system

2.3. The fuzzy inference system model

The rule structure and parameters of the fuzzy inference system are automatically generated using the process values. The Adaptive Neuro-Fuzzy Inference system (ANFS) algorithm is adopted for system construction. The fuzzy clustering algorithm uses an objective function to determine the number of the rules. The fuzzy partition matrix U and the center matrix V is can be defined as follows:

$$U=\begin{pmatrix} U_{11} \cdots U_{1k} \cdots U_{1N} \\ \vdots \\ U_{i1} \cdots U_{ik} \cdots U_{iN} \\ \vdots \\ U_{c1} \cdots U_{ck} \cdots U_{cN} \end{pmatrix}$$

$$V=\{v_1, v_2, \cdots v_c\}\ , v_i \in R^m \qquad (5)$$

In this formula, c(2.c.N) is the number of clustering groups, μ_{in} is the membership degree showing that x_k is included to the i-th clustering group, and v_i shows the n-dimensional center vector element. To determine the values μ_{in}, v_i to minimize the above objective function, the values can be expressed as follows:

$$u_{ik}=\frac{1}{\sum_{j=1}^{c}\left(\frac{d_{ik}}{d_{jk}}\right)^{\frac{2}{m-1}}}$$

$$v_{il}=\frac{\sum_{k=1}^{N}(u_{ik})^m x_{kl}}{\sum_{k=1}^{N}(u_{ik})^m}\ , l=1,\cdots,n \qquad (6)$$

These values can be obtained through iterative calculations. The number of the rules is determined and the characteristic membership function values of the rules are generated using the neural network. The ANFIS model shown in Figure 5 uses fuzzy variables in the neural network.

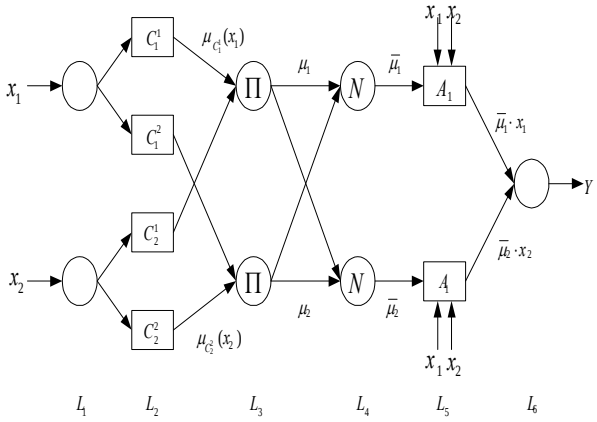


Figure 5. The ANFIS model

The ANFIS model in Figure 5 shows a network-type structure similar to that of a neural network. The network maps inputs through input membership functions and various characteristic values, and then through output parameters to outputs. The model can be used to interpret the input/output map that has the same meaning when compared with the original fuzzy inference system. The overall system is composed of six levels. The first level transfers the input data to the next level. The fuzzy variable of the second level is the same as that of the premise part of the fuzzy model. That is, the characteristic values of the fuzzy variables are learned through iterative calculations. The nodes of the third level output the compatibility values after calculating the multiplication and addition of the previous input data. In level 4, the data is normalized. The fuzzy variables of level 5 are the same as the result of the fuzzy model. The parameters of linear combination are then learned.

Figure 6 shows the overall flow chart for the iterative calculations where the learning data is used to determine the number of the rules. The constant C is

the initial rule number and the next process determines the parameters of the premise and consequence part of the rules.

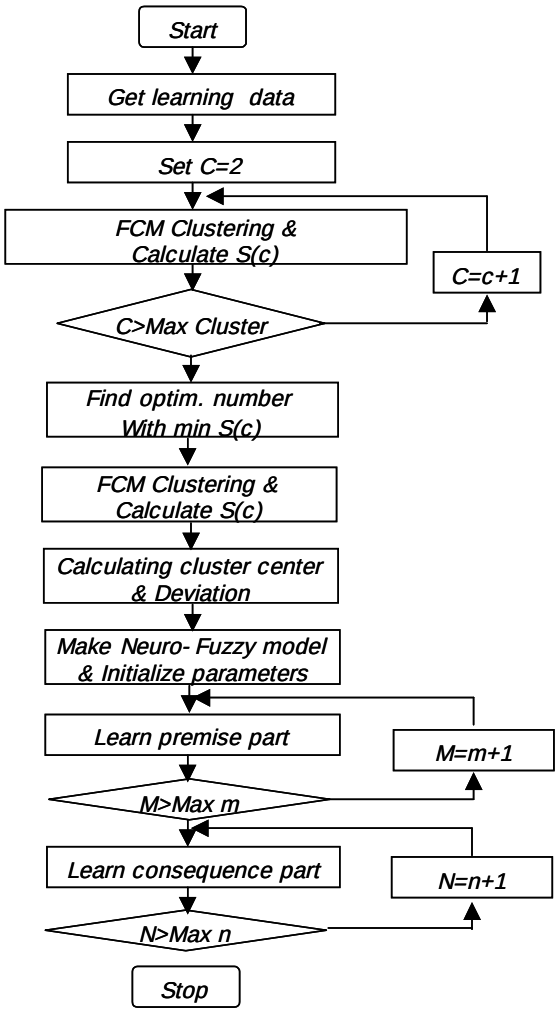


Figure 6. The flow for the modeling for the fuzzy inference system

3. Fuzzy controller implementation

The fuzzy process controller is implemented as the function block diagram in the DCS system. The output and performance of the fuzzy controller is verified using MATLAB. Figure 7 shows the definition diagram for parameters in the fuzzy controller. After determining the structure and the various parameters of the fuzzy control system, the operator/engineer types the relevant data into input forms. The input data includes the number of input/output data, the number of fuzzy rules, the number of the membership functions, the minimum /maximum values of the process input data, the characteristic values of each membership function, and the parameters of the output. Figure 8 shows the download function of the defined parameters. The data are downloaded to the designated real time control station.

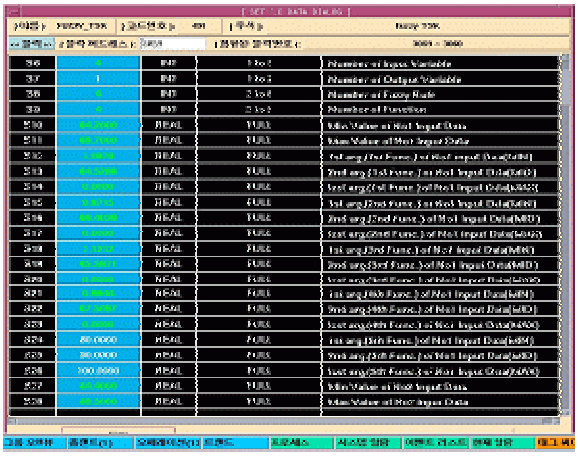


Figure 7. The definition function block diagram

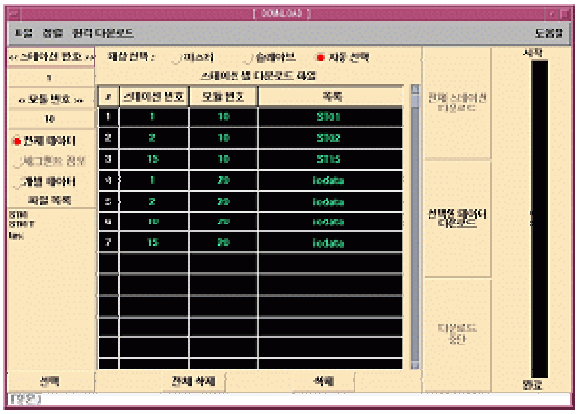
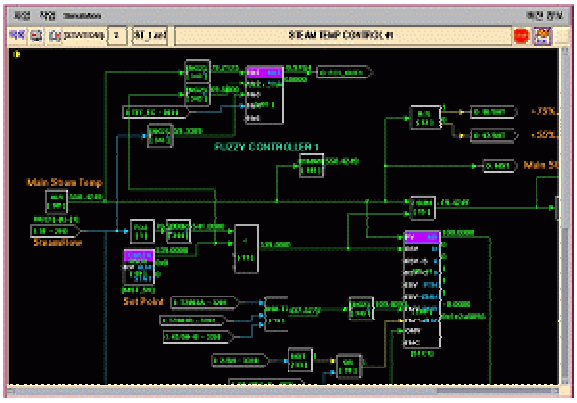


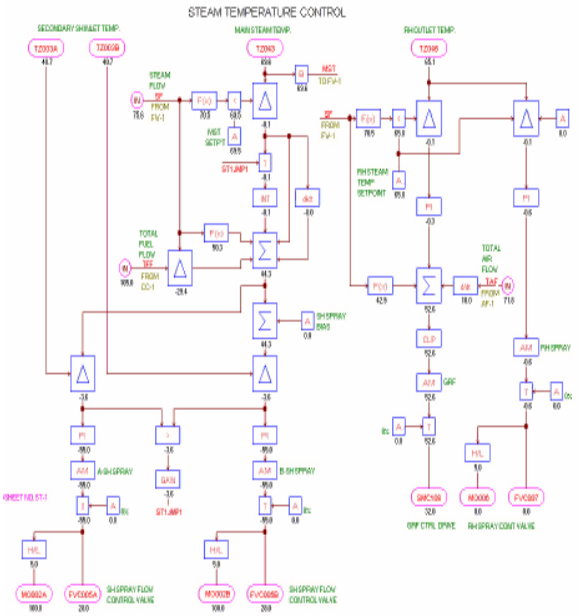
Figure 8. The download function of the parameters

4. Experimental results

The experimental system is composed of the DCS system and the simulator system. The DCS system is responsible for the control and the simulator corresponds to the real process. The following Figure 9 shows the experimental configuration for the SH steam temperature control system. Figure 9(a) shows the fuzzy controlled function block diagram in the DCS and Figure 9(b) represents the process block diagram in the simulator.



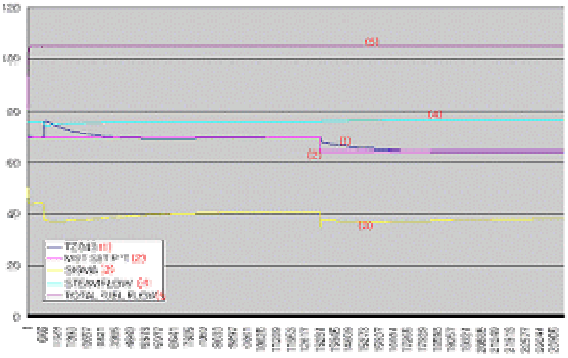
(a) The control function block diagram of DCS



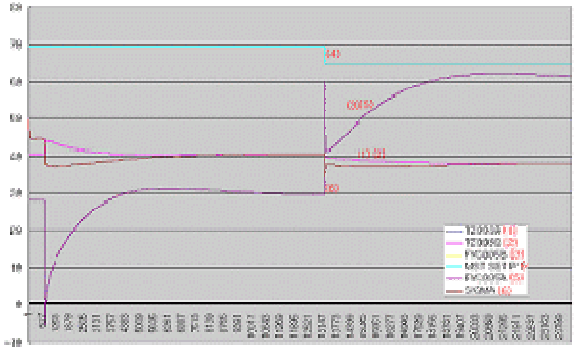
(b) The simulator process

Figure 9. The experimental system configuration

Figure 10 shows the process characteristics in the simulator. The responses of the closed loop system are shown by the changes of the operating points between 540˚C and 529˚C. The system represents the first order system and includes a large time constant, where the settling time is about 450 seconds. The data are gathered in the simulator's file system and the parameters of the rules for the two fuzzy controllers are obtained using a fuzzy clustering algorithm and the previous data. Figure 11 shows the input/output spaces for the fuzzy controller. They represent the nonlinear and smoothing characteristics of the relations between the input and the output.

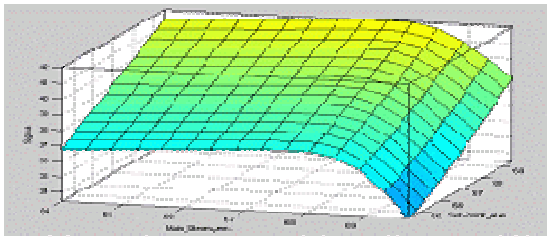


(a) The main control part

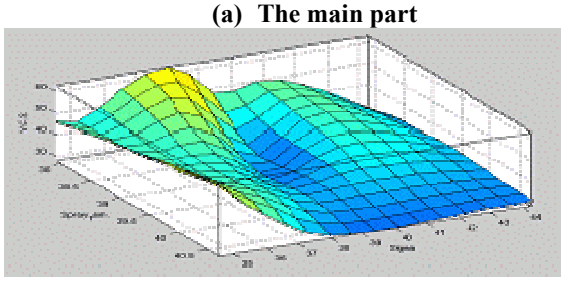


(b) The sub control part

Figure 10. The simulator system response



(a) The main part



(b) The sub part

Figure 11. The fuzzy controller input-output space

Figure 12 shows the result of the case when applying the fuzzy controller in the DCS where the rapid response may be compared with the simulator control system and obtain a settling time of the about 150 seconds.

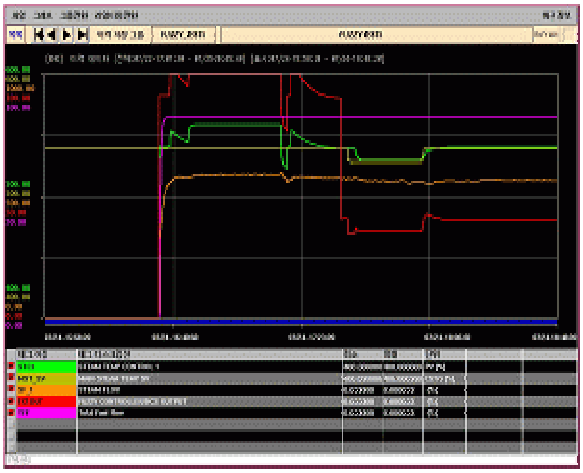


Figure 12. Fuzzy controller responses

5. Conclusion

In this paper, a fuzzy controller for a superheated steam temperature process controller was developed using operating expertise and robust experimental numeric control data and the controller was implemented in the function block of the DCS. Experimental analysis was accomplished through simulation of the DCS and implementation of the fuzzy control system. The result has shown good performance by the proposed fuzzy controller system and verifies the applied knowledge engineering approach. These results indicate that the fuzzy controller system is suitable for real system integration and testing.

Acknowledgements

This research was supported by the MIC(Ministry of Information and Communication), Korea under the ITRC(Information Technology Research Center) support program supervised by the IITA(Institute of information Technology Assessment) IITA-2006-C1090-0603-0024.

6. References

[1] Byung-so cho, Chae-Joo Moon, "Design of Fuzzy Seawater lift pump system in Fossil fired power plant", Proceedings of ISUMA-NAFIPS'95, p.204-208, 1995.

[2] D. H Kim, "Power Plant Boiler Level Control using Immune Algorithm based Fuzzy Rules Auto-tuning", The IASTED Conference on Artificial Intelligence, 2004, p. 201-208.

[3] A. Diaz, M. Jimenez and M. Strefezza," Implanting a Feedforward Fuzzy Controller to a Pilot Tunk Plant", Intelligent Systems and Control, Vol. 366, No. 1, 2002.

[4] Hamed Peyravi, Abdollah Khoel and Khayrollah Hadid, " Design of an analog CMOS fuzzy logic controller chip", Fuzzy sets and Systems, Vol.132, No 2. p. 245-260, 2002.

[5] N. Hossein-Zadeh, A Kalam, "On-line tuning of a fuzzy-logic power system stabilizer", Journal of Science and Technology, Vol. 26, No. B4, 2002.

[6] Flores. Araya. J, Berenguel, "Fuzzy Predictive Control of a Solar Power Plant", Fuzzy Systems, Vol. 13, No.1, p. 58-68, 2005.

Un Acercamiento a los Modelos Multidimensionales Espacio Temporales

Francisco Javier Moreno Arboleda
Facultad de Minas, Escuela Sistemas,
Universidad Nacional de Colombia,
Sede Medellín.
fjmoreno@unal.edu.co

Fernando Arango Isaza
Facultad de Minas, Escuela de Sistemas,
Universidad Nacional de Colombia,
Sede Medellín.
farango@unal.edu.co

Resumen

En este artículo se presenta un recorrido por los principales modelos multidimensionales espacio temporales reportados hasta la fecha. Se propone un conjunto de requisitos deseables y aplicables a este tipo de modelos con el fin de evidenciar el grado de contribución de cada trabajo examinado. El objetivo es establecer el nivel de desarrollo y presentar oportunidades de investigación en este campo de trabajo.

1. Introducción

Desde 1996 con el surgimiento y desarrollo de las bodegas de datos [1,2] ha habido un enorme interés por los modelos multidimensionales. En los últimos años el interés se ha centrado en el enriquecimiento de dichos modelos con aspectos espaciales y temporales. Con el fin de establecer la utilidad de los modelos examinados se propone un conjunto de requisitos deseables. El objetivo final es presentar oportunidades de investigación en este campo y establecer su nivel actual de desarrollo.

El contenido del artículo es el siguiente: en la Sección 2 se presenta el marco teórico, la Sección 3 define los requisitos bajo los cuales serán examinados los diferentes modelos que se describen en la Sección 4. Finalmente, en la Sección 5 se presenta una discusión del tema y posibles trabajos futuros.

2. Conceptos básicos

Un modelo multidimensional, como su nombre lo sugiere, consiste de un conjunto de *dimensiones* que son asociadas a un fenómeno medible de interés para una organización. Este fenómeno es denominado *hecho*. Las dimensiones se componen de *niveles* los cuales se estructuran en jerarquías estableciendo un orden parcial. La Figura 1 muestra una dimensión geográfica con niveles ciudad, estado, región y país. Tradicionalmente las dimensiones han sido tratadas de forma textual (alfanumérica) y las medidas de un hecho de forma numérica.

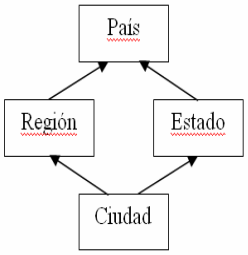


Figura 1. Ejemplo de dimensión geográfica

Una revisión de modelos multidimensionales de este tipo puede verse en [3].

En los últimos años los modelos multidimensionales se han enriquecido con elementos traídos de las bases de datos espaciales y temporales. Una base de datos espacial está orientada al soporte de geometrías (puntos, líneas, regiones) asociadas a objetos [4]. Una base de datos temporal extiende el conocimiento almacenado en una base de datos acerca del estado actual del mundo incluyendo el pasado [5]. Por lo tanto, una base de datos espacio temporal soporta geometrías que cambian con el tiempo, ya sea en forma o posición [6]. La presencia de espacialidad y temporalidad en un modelo multidimensional se puede reflejar tanto en las dimensiones como en los hechos tal y como se expone en los trabajos examinados en la Sección 4.

Otro concepto de interés es el de *inclusión*. Considérese por ejemplo los niveles ciudad y estado de la Figura 1. Sean *c1* y *e1* instancias de estos niveles respectivamente. Se dice que *c1* está incluida en *e1* si

existe una relación de pertenencia (ya sea lógica, geográfica, administrativa etc.) que indica que *c1* está asociada a *e1*. Esta inclusión se denominará *total* si *toda* la instancia *c1* está asociada a *e2*. Recientemente en [7] se generalizó este concepto y se estableció el de *inclusión parcial.* Como ejemplo considérense los niveles carretera y ciudad tal y como se muestra en la Figura 2a. Una carretera *r1* puede estar incluida parcialmente en una ciudad *c1*, tal y como lo muestra la Figura 2b.

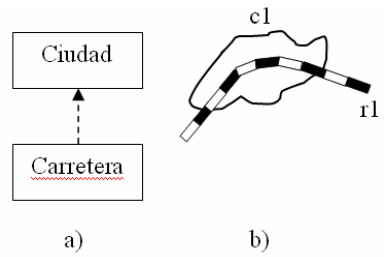


Figura 2. a) Inclusión parcial entre niveles b) Inclusión parcial en las instancias

3. Requisitos deseables en un modelo multidimensional espacio temporal

Con el fin de evaluar diversos modelos, se procederá a establecer un conjunto de requisitos deseables para tal efecto. Dichos requisitos están orientados a establecer los aspectos de expresividad y facilidades de manipulación (concepción de consultas y definición del esquema) de cada modelo.

3.1. Inclusión (I)

Se refiere al tipo de inclusión soportada por el modelo, tal y como se describió en la Sección 2.

3.2. Espacialidad (E)

Se refiere a los aspectos de espacialidad que son soportados por el modelo. Se puede considerar desde dos puntos de vista: tipos de elementos espaciales soportados (puntos, líneas, regiones etc.) y presencia de estos elementos en los hechos y en los niveles de las dimensiones.

3.3. Temporalidad

Se refiere al tipo de temporalidad soportada por el modelo. Ésta se puede reflejar en tres aspectos:

3.3.1. Temporalidad en las instancias (TI). Se refiere a preservar la evolución de la asociación entre las instancias de los niveles de una dimensión, por ejemplo, sean los niveles vendedor y ciudad. Un vendedor *v1* puede estar asociado a una ciudad *c1* durante un período *p1* y en un período *p2* puede estar asociado a una ciudad *c2*.

3.3.2. Temporalidad en el esquema (TE). Se refiere a preservar la evolución de la estructura de una dimensión. Como ejemplo, considérese una dimensión conformada por los niveles ciudad y país durante un período *p1*. En un período *p2* un nivel *región* es insertado entre los niveles anteriores. Asimismo, en un período *p3* el nivel país podría desaparecer.

3.3.3. Versionamiento (V). Se refiere a la capacidad para recuperar la información de acuerdo a las diferentes evoluciones (versiones) que ha sufrido el esquema. Por ejemplo, proyectar la información correspondiente a un esquema actual en un esquema anterior, proyectar la información de un esquema anterior en un esquema actual.

3.4. Espacialidad y temporalidad (E+T)

Se refiere al soporte de la evolución de los elementos espaciales. Como ejemplo, considérese un nivel ciudad donde se conserva la evolución del crecimiento/decrecimiento de la región (croquis) correspondiente a cada ciudad.

3.5. Lenguaje

Se analiza desde dos aspectos:

3.5.1. Lenguaje de definición del esquema (LE). Se refiere a si se ofrece algún formalismo para crear los elementos del esquema correspondientes al modelo propuesto.

3.5.2. Lenguaje de consulta (LC). Se refiere a si se define algún formalismo para el planteamiento de consultas dentro del modelo propuesto.

3.6. Agregación espacial (AE)

Tradicionalmente los hechos contienen medidas numéricas, las cuales son agregadas, por defecto, mediante una operación suma. Sin embargo, si se presenta un hecho con una medida espacial, surge el interrogante: ¿Cómo debe abordarse la agregación de tales medidas?

4. Modelos examinados

El foco de interés de esta sección son los modelos multidimensionales que tienen soporte espacial, temporal o espacio temporal. Los modelos reportados serán analizados de acuerdo a los requisitos presentados en la Sección 3. Este análisis permite evidenciar las fortalezas y debilidades de cada propuesta. A su vez permite descubrir carencias que pueden dar lugar a trabajos futuros tal y como se discute en la Sección 5.

4.1 Modelos multidimensionales con aspectos espaciales

[8] propone un modelo donde tanto las dimensiones como las medidas pueden ser espaciales y no espaciales. Las dimensiones pueden ser no espaciales, semiespaciales (algunos niveles espaciales) y totalmente espaciales (todos los niveles espaciales.) Una medida espacial es definida como un conjunto de punteros a objetos espaciales.
Su trabajo se enfoca en presentar algoritmos para implementar eficientemente la agregación espacial de las medidas espaciales, mediante la operación *merge* [9]. Para ello realiza una extensión a los algoritmos (no espaciales) de [10].
[11] adopta la definición de dimensión espacial de [8] y se concentra en la implementación para la solución de consultas de tipo rango espacial (*window query*) por medio de una modificación del algoritmo de búsqueda GiST [12] y uso de árboles R. Por ejemplo, considérese la consulta "obtener el total de ventas de todas las estaciones de gas dentro de una región rectangular dada". Acá la dimensión espacial está constituida por las estaciones de gas.
Otro de sus aportes consiste en la búsqueda de respuestas de agregación aproximadas, un problema poco trabajado y reportado en [13]. Los autores proponen realizar un manejo más apropiado de estas aproximaciones, incorporando diversas distribuciones de probabilidad y un tratamiento formal del error.
[14] propone un modelo multidimensional espacial denominado Spatial MultiDimER. Extiende la notación gráfica del modelo Entidad-Relación [15] y de MADS [16], un modelo conceptual espacio temporal (no multidimensional.)
Extiende el trabajo de [8] en cuanto a la definición de las dimensiones y medidas espaciales. Por ejemplo una medida espacial puede ser:
- Una geometría *acompañada* de una función de agregación espacial, por ejemplo unión geométrica.
- Un valor numérico derivado a partir de operaciones o relaciones espaciales, por ejemplo, una medida *mínima distancia* podría ser derivada para un hecho que tiene relación con 2 objetos espaciales.

[17] utiliza la notación propuesta en [14] y clasifica las jerarquías espaciales en simétricas, asimétricas y generalizadas.
Una jerarquía es *simétrica* si está conformada por un único camino entre el nodo raíz y el nodo hoja. En las instancias de la jerarquía todos los niveles son obligatorios. Si algunos niveles son opcionales, la jerarquía es *asimétrica*.
Una jerarquía es *generalizada* si existen varios caminos *excluyentes* entre el nodo raíz y el nodo hoja. Los caminos pueden compartir niveles. Cada instancia de la jerarquía pertenece a un solo camino.
También se examinan las jerarquías *no estrictas*, es decir, cuando hay relaciones muchos a muchos entre un par de niveles [18, 7] y las jerarquías con múltiples caminos no excluyentes.
Finalmente, se analizan otros tipos de relaciones espaciales [19] posibles entre niveles espaciales, como *within* (inclusión total), *intersects* u *overlaps* (inclusión parcial) *touches* entre otras; y cómo éstas influyen en el proceso de agregación de las medidas.
[20] propone un modelo multidimensional espacial cuyo principal aporte lo constituye el manejo de medidas complejas y sus correspondientes aspectos de agregación. Una medida compleja es aquella que se compone de aspectos espaciales y no espaciales (alfanuméricos.) Los autores permiten que el usuario especifique cómo se debe agregar una medida compleja por medio de un *modo de agregación*.
[7] presenta un modelo multidimensional espacial con soporte de inclusión parcial.
Los autores muestran que la inclusión parcial es un caso general de la inclusión total [3]. Se presenta también un lenguaje de consulta con un enfoque algebraico el cual incluye operaciones de selección, unión, agregación entre otras.
En [21] se realiza una extensión probabilística al modelo de [7]. La idea es que en el modelo preliminar, el manejo de la inclusión parcial adopta una aproximación "segura", es decir, si se sabe que un hecho *h* sucede en una carretera *c* y dicha carretera tiene un porcentaje contenido en la ciudad *z*, no se puede asegurar que el hecho *h* haya sucedido en la ciudad *z*. La idea es utilizar los porcentajes de inclusión como indicadores de probabilidad, es decir, se puede decir que el hecho *h* sucedió con una probabilidad *p* en una determinada ciudad. Los autores utilizan este aspecto para incorporarlo en el lenguaje de consulta que proponen.
[22] propone un método para integrar modelos multidimensionales (sin espacialidad), a los cuales denomina bases de datos estadísticas (SDB) con bases de datos geográficas objetuales (OGDB). La idea es

establecer un mecanismo de asociación entre las clases geográficas de una OGDB y las variables (niveles) geográficas de una SDB. Ésto permite responder consultas que requieren de ambas bases de datos, es decir, utilizar las capacidades analíticas de una SDB junto con las capacidades espaciales y objetuales de una OGDB.

4.2 Modelos multidimensionales con aspectos temporales

[23] propone un modelo multidimensional temporal sin elementos espaciales. El modelo preserva la evolución tanto a nivel de las instancias como de la estructura de las dimensiones. Se propone además un lenguaje de consulta, TOLAP, que permite expresar consultas como ¿Cuál fue el total de ventas del vendedor *v1* cuándo estaba asignado al almacén *a1*? ¿Poseía la dimensión geográfica el nivel región en una fecha específica?
Sin embargo, su trabajo no aborda aspectos de versionamiento a diferencia del de [24] que aborda el versionamiento *entre instancias*, mediante funciones de transformación que permiten presentar los resultados en una determinada versión. Sin embargo, no se maneja versionamiento a nivel del esquema. En ese sentido los dos trabajos se complementan.
[25] propone un modelo que maneja versionamiento tanto a nivel de instancias como de esquema. Se introducen las nociones de *modos temporales de presentación* y de *tabla de hechos multiversión*. De acuerdo a las diferentes versiones estructurales se puede utilizar la versión correspondiente de la tabla de hechos. También se proponen operadores para facilitar el versionamiento: insertar una versión de un miembro, (por ejemplo, un miembro distrito D1, válido durante un período p, puede dividirse en dos miembros D11 y D12), eliminar una versión de un miembro, reclasificarlo en la jerarquía dimensional, entre otros.

4.3 Modelos multidimensionales con aspectos espacio temporales

Son pocos los modelos multidimensionales encontrados que manejen tanto el aspecto espacial como el temporal.
Los trabajos encontrados son bastante incipientes, carecen de tratamiento formal y presentan sólo ideas preliminares sobre el tipo de problemas que podrían ser abordados, en la Sección 5 se discute más al respecto.
[26] presenta un caso de estudio de un modelo multidimensional con elementos espacio temporales. Utiliza Perceptory [27], el cual es una notación gráfica para modelado espacio temporal. Aborda el problema del manejo de jerarquías dinámicas en las dimensiones espaciales, es decir, la conformación de una jerarquía no siempre puede predefinirse en tiempo de diseño.
Este problema es mencionado en [13] y tratado en [28] pero sin considerar espacialidad.
Sin embargo, la solución propuesta en [26] es un modelo particular, no se generaliza, ni se formaliza. Tampoco se propone un lenguaje de consulta para la manipulación de su modelo.
Se menciona, pero no se resuelve, el problema de la concepción de consultas que a su vez retornan relaciones espacio temporales.
[29] presenta un caso de estudio que integra modelos multidimensionales espaciales correspondientes a un bosque tomados en diferentes épocas. Debido a que los datos fueron obtenidos en diferentes épocas (períodos de 10 años) surgen problemas de estandarización, por ejemplo, en una década la edad de los árboles se clasificaba de forma cualitativa y en otra de forma cuantitativa. Se propone entonces la creación de clasificaciones unificadas para el modelo integrador. Sin embargo, la técnica de integración propuesta no se generaliza ni formaliza. Tampoco se propone un lenguaje especializado de consulta para manipular el modelo.
En la Tabla 1 se realiza una comparación de los modelos analizados frente a los requisitos propuestos.

Tabla 1. Modelos vs. requisitos analizados

Modelos		Requisitos								
		I	E	TI	TE	V	E+T	LE	LC	AE
	[8]	Total	Sí	No	No	No	No	No	No	Sí
	[11]	Total	Sí	No	No	No	No	No	No	Sí
	[14,17]	Total	Sí	No	No	No	No	No	No	Sí
	[20]	Total	Sí	No	No	No	No	No	No	Sí
	[7]	Parcial	Sí	No	No	No	No	No	Sí	Sí
	[21]	Parcial	Sí	No	No	No	No	No	Sí	Sí
	[22]	Total	Sí	No	No	No	No	Inc.	Inc.	Sí
	[23]	Total	No	Sí	Sí	Inc.	No	No	Sí	No
	[24]	Total	No	Sí	No	Sí	No	Inc.	Inc.	No
	[25]	Total	No	Sí	Sí	Sí	No	Inc.	No	No
	[26]	Total	Sí	No	Inc.	No	Inc.	No	No	No
	[29]	Total	Sí	Inc.	No	No	Inc.	No	No	No

5. Discusión y Conclusiones

En los últimos años ha habido mucho trabajo en el área de las bases de datos espacio temporales [6], también conocidas como bases de datos de objetos móviles.
Los campos de aplicación incluyen análisis entre otras:
- Demográfico (desplazamiento de poblaciones.)
- Ecológico (crecimiento/decrecimiento de bosques.)
- Mercadeo (análisis del movimiento de vendedores.)
- De fenómenos naturales (movimientos de huracanes.)
- Militar (movimiento de tropas, regiones ocupadas.)
- Urbanístico (crecimiento/decrecimiento de ciudades.)

De otro lado, los modelos multidimensionales también han mostrado sus bondades gracias a las facilidades que ofrecen para realizar análisis de grandes volúmenes de datos por medio de herramientas OLAP [30].

La unión de estos dos mundos es bastante prometedora, sin embargo las propuestas que realizan tal "fusión" son pocas y su aproximación aún es incipiente. La mayoría de los trabajos se concentra en el aspecto espacial o en el temporal pero no en ambos.

En cuanto a trabajos que se propone abordar están:

- Problemas que aún persisten en los modelos multidimensionales que no poseen elementos espaciales ni temporales. Un reporte al respecto puede verse en [13]. Se identifican problemas como: manejo de metadatos, agregación aproximada (situaciones en las que una respuesta aproximada es suficiente y la precisión no es indispensable), versionamiento, seguridad, entre otros.
- Operaciones de cruce entre modelos multidimensionales espacio temporales, es decir, consolidar datos provenientes desde diferentes modelos, operación conocida como drill across [2] en OLAP. Como puntos de partida véase [31, 32].
- Tal y como se expone en [17], para la relación espacial entre niveles espaciales se ha trabajado básicamente con inclusión (overlaps). Se deben explorar otras posibilidades como intersección, disyunción, adyacencia etc.
- Desarrollo de lenguajes especializados tanto para consultar como para definir esquemas para modelos multidimensionales espacio temporales.
- Explotar el modelo objetual en los modelos multidimensionales espacio temporales. Como puntos de partida véase [22,33].
- Finalmente, la agregación de medidas espaciales requiere más trabajo: algoritmos eficientes y exploración de más operaciones espaciales, como intersección y solapamiento, para llevar a cabo tal agregación. Como punto de partida véase [34].

Agradecimientos: Este trabajo se desarrolla dentro del marco del Doctorado en Ingeniería de Sistemas de la Universidad Nacional de Colombia auspiciado por Colciencias.

6. Referencias

[1] Inmon W.H., *Building the Data Warehouse*, John Wiley & Sons, Nueva York, 2005.

[2] S. Chaudhuri, and U. Dayal, "An Overview of Data Warehousing and OLAP Technology", *ACM SIGMOD Record*, Marzo 1997, pp. 65-74.

[3] T.B. Pedersen, and C.S. Jensen, "Multidimensional Data Modeling for Complex Data", *Proceedings of the ICDE '99*, Sidney, 1999, pp. 336-345.

[4] R.H. Güting, "An Introduction to Spatial Database Systems", *VLDB Journal 3(4)*, 1994, pp. 357-399.

[5] Date C.J., H. Darwen, and N. Lorentzos, *Temporal Data & the Relational Model*, Morgan Kaufmann, San Francisco, 2002.

[6] Güting R.H., and M. Schneider, *Moving Objects Databases*, Morgan Kaufmann, San Francisco, 2005.

[7] C.S. Jensen, A. Kligys, T.B. Pedersen, and I. Timko, "Multidimensional Data Modeling for Location-Based Services", *VLDB Journal 13(1)*, 2004, pp. 1-21.

[8] J. Han, N. Stefanovic, and K. Koperski, "Selective Materialization: An Efficient Method for Spatial Data Cube Construction", *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'98)*, Melbourne, 1998, pp. 144-158.

[9] Shekhar S., and S. Chawla, *Spatial Databases: A Tour*, Prentice Hall, 2003.

[10] V. Itarinarayan, A. Rajaraman, and J. D. Ullman, "Implementing Data Cubes Efficiently", *Proceedings of 1996 ACM-SIGMOD International Conference Management of Data*, Montreal, pp. 205-216.

[11] F. Rao, L. Zhang, X. Yu, Y. Li, and Y. Chen, "Spatial Hierarchy and OLAP-Favored Search in Spatial Data Warehouse", *DOLAP 2003*, Nueva Orleans, 2003, pp. 48-55.

[12] J.M. Hellerstein, J.F. Naughton, and A. Pfeffer, "Generalized Search Trees for Database Systems", *VLDB 1995*, Zurich, pp. 562-573.

[13] W. Hümmer, W. Lehner, A. Bauer, L. Schlesinger, "A Decathlon in Multidimensional Modeling: Open Issues and Some Solutions", *DaWaK 2002*, Aix-en-Provence, 2002, pp. 275-285.

[14] E. Malinowski, and E. Zimanyi, "Representing Spatiality in a Conceptual Multidimensional Model", *GIS 2004*, Washington D.C, pp. 12-22.

[15] P.P. Chen, "The Entity-Relationship Model - Toward a Unified View of Data", *ACM Transactions on Database Systems (TODS) 1(1)*, 1976, pp. 9-36.

[16] C. Parent, S. Spaccapietra, and E. Zimanyi, "Spatio-Temporal Conceptual Models: Data Structures + Space + Time", *ACM-GIS '99*, Kansas, 1999, pp. 26-33.

[17] E. Malinowski, and E. Zimanyi, "Spatial Hierarchies and Topological Relationships in the Spatial MultiDimER Model", *BNCOD 2005*, Sunderland, 2005, pp. 17-28.

[18] I. Song, W. Rowen, C. Medsker, and E. F. Ewen, "An Analysis of Many-to-Many Relationships Between Fact and Dimension Tables in Dimensional Modeling", *DMDW 2001*, Interlaken, 2001, pp. 6-13.

[19] N. Tryfona, and M. Egenhofer, "Consistency Among Parts and Aggregates: a Computational Model", *Transactions in GIS 1(3)*, 1997, pp. 189-206.

[20] S. Bimonte, A. Tchounikine, and M. Miquel, "Towards a Spatial Multidimensional Model", *DOLAP 2005*, Bremen, 2005, pp. 39-46.

[21] I. Timko, C.E. Dyreson, and T.B. Pedersen, "Probabilistic Data Modeling and Querying for Location-Based Data Warehouses", *SSDBM 2005*, Santa Barbara, pp. 273-282.

[22] F. Ferri, E. Pourabbas, M. Rafanelli, and F.L. Ricci, "Extending Geographic Databases for a Query Language to Support Queries Involving Statistical Data", *SSDBM 2000*, Berlín, 2000, pp. 220-230.

[23] A.O. Mendelzon, and A.A. Vaisman, "Temporal Queries in OLAP", *VLDB 2000*, pp. 242-253.

[24] J. Eder, and C. Koncilia, "Changes of Dimension Data in Temporal Data Warehouses", *DaWaK 2001*, Munich, 2001, pp. 284-293.

[25] M. Body, M. Miquel, Y. Bèdard, and A. Tchounikine, "A Multidimensional and Multiversion Structure for OLAP Applications", *DOLAP 2002*, McLean, 2002, pp. 1-6.

[26] G. Pestana, and M. Mira da Silva, "Multidimensional Modeling Based on Spatial, Temporal and Spatio-Temporal Stereotypes", *ESRI International User Conference*, San Diego, 2005, pp. 1-11.

[27] Y. Bèdard, S. Larrivée, M.J. Proulx, and M. Nadeau, "Modeling Geospatial Databases with Plug-Ins for Visual Languages: A Pragmatic Approach and the Impacts of 16 Years of Research and Experimentations on Perceptory", *ER 2004*, Shanghai, 2004 pp. 17-30.

[28] C.A. Hurtado, A.O. Mendelzon, and A.A. Vaisman, "Updating OLAP Dimensions", *DOLAP 1999*, Kansas City, 1999, pp. 60-66.

[29] M. Miquel, Y. Bèdard, A. Brisebois, J. Pouliot, P. Marchand, and J. Brodeur, "Modeling Multidimensional Spatio-temporal Data Warehouse in a Context of Evolving Specifications", *Joint International Symposium International Society for Photogrammetry and Remote Sensing (ISPRS) Commission IV SDH 2002 95th Annual CIG Conference*, Otawa, 2002, pp. 1-6.

[30] Kimball R., and M. Ross, *The Data Warehouse Toolkit: the Complete Guide to Dimensional Modeling*, John Wiley & Sons, Nueva York, 2002.

[31] M. Golfarelli, D. Maio, and S. Rizzi, "The Dimensional Fact Model: A Conceptual Model for Data Warehouses", *IJCIS 7(2-3)*, 1998, pp. 215-247.

[32] A. Abelló, J. Samos, and F. Saltor, "On Relationships Offering New Drill-Across Possibilities", *DOLAP 2002*, Mclean, 2002, pp. 7-13.

[33] F. Ravat, and O. Teste, "A Temporal Object-Oriented Data Warehouse Model", *DEXA 2000*, Londres, 2000, pp. 583-592.

[34] I.F. Vega, R. Snodgrass, and B. Moon, "Spatiotemporal Aggregate Computation: A Survey", *IEEE Transactions on Knowledge and Data Engineering 17(2)*, Febrero 2005, pp. 271-286.

Asynchronous Merging of Software Ontologies: An Experience

Nicolas Anquetil[1], Aurora Vizcaíno[2], Francisco Ruiz[2], Kathia Oliveira[1] and Mario Piattini[2]
[1] *GES Research Group. Catholic University of Brasilia, Brazil*
{anquetil, kathia}@ucb.br
[2] *Alarcos Research Group. University of Castilla-La Mancha, Spain*
{aurora.vizcaino, francisco.ruiz, mario.piattini}@uclm.es

Abstract

Different methodologies exits to merge ontologies. However, most of them need the source ontologies to be defined in a particular and formal way. Moreover, tools to help during the merging process have been developed, but these are thought for synchronous settings (where the knowledge engineers can exchange ideas in real time) and very specific conditions. In this paper we describe the merging process that two research groups have used to merge two maintenance ontologies in an asynchronous way. We also describe the problems that we faced since each research group was in a different country, with different time zone and also different mother languages. The usage of a systematic methodology helped us to tackle these problems as will be explained in this paper.

1. Introduction

Onotologies capture consensual knowledge of a specific domain in a generic and formal way, to allow it to be reused and shared among groups of people. Despite requiring consensus between different experts, there is no single possible ontology to model a particular domain, thus domain-specific ontologies are modeled by multiple authors in multiple settings [24]. For example, in the case of software engineering where ontologies can play important roles, and more concretely in the software maintenance domain, there are several published ontologies [3, 4, 14, 22], each one dealing with maintenance activity from a different point of view. In an attempt to achieve a better result we decided to merge two ontologies, those of Dias *et al*. [4] and Ruiz *et al*. [22] that seemed to be most complementary and which moreover, were based on a third, that of Kitchenham *et al.* [14]. Our goal was to construct a more general ontology by taking into account the most important concepts related to software maintenance.

A great difficulty in this work was the geographic distance between the two teams of authors of the ontologies. As a result, the merging could not be conducted in a typical way where the knowledge engineers could meet and discuss together what concepts to include, what restrictions to apply to these concepts, etc. What is more, we had some extra challenges. For instance, one ontology was developed by a Brazilian University, and the other by a Spanish University and although both ontologies were defined in English this was not the mother tongue of any of the developers of the ontologies. Because of this, misunderstandings might arise. We attempted to counter balance these difficulties by defining a merging process that would take the specificity of our situation into account.

In this paper we present the process followed to merge the two ontologies and we report on the difficulties found and the lessons learned from this experiment.

The remainder of the paper is structured as follows. Section 2 describes the merging process and some methodologies and tools developed for this purpose. Section 3 explains the process that we followed to merge the ontologies. Section 4 presents the benefits and limitations of the approach used. Finally in section 5 conclusions are outlined.

2. Merging ontologies

It is first necessary to clarify the difference between two related words: merging and alignment. Merging ontologies means to create a single coherent ontology from two sources. Aligning ontologies means to establish links between them and allow them to reuse information from one another [16]. Alignment does not aim to create a new ontology. Merging two ontologies

implies some kind of alignment as one must map one ontology on to the other to find out the commonalities, synonyms, etc.

There are different methods by which to merge ontologies and many of them provide a tool with which to automatically identify potential matchings or provide an environment to manually find and define these matchings. Mapping tools and algorithms are: ONIONS [23] which allows the creation of a library of ontologies originating from different sources; the Chimaera system [15] provides support to merge ontological terms from different sources, to check the coverage and correctness of ontologies and to maintain ontologies over a period of time; OntoMorph [2] provides two kinds of mechanisms for merging ontologies. One is a syntactic rewriting support that allows translation between two different representation languages, and the other is a semantic rewriting tool that allows inference-based transformations; GLUE [5] uses machine learning techniques, to provide pairs of related concepts with some certainty factor associated to each pair. Another approach is FCA-Merge [24] which takes as input two ontologies to be merged and a set of documents on the domain of the ontologies. The merging is performed by extracting instances that belong to concepts of both ontologies from the documents. Finally, PROMPT is an algorithm embedded in Protégé 2000, that proposes first to elaborate a list with the operations to be performed in order to merge two ontologies [17]. This activity is carried out automatically by a PROMPT plug-in. Then, an iterative process is performed. For each iteration the ontology developer selects an operation of the list and executes it. After that, a list of conflicts is generated and the list of possible operations for the following iterations is updated.

Most previous techniques need the source ontologies to be defined in a particular and formal way and some, such as OntoMorph and Chimarea, use a description logics based approach. Moreover, only FCA-MERGE offers a structural description of the global merging process [24]. These facts, and other difficulties that will be detailed in the next section, led us to define our own merging approach. Contrary to the existing approaches, we did not seek automation of the merging process and will not propose any tool to help. In our experience, very few activities can be automated and when this is possible, they do not represent a significant work load. We will therefore focus on presenting and discussing our methodology which has given good results and proved to be useful.

3. Merging two software maintenance ontologies

This work started as a result of two teams (the Alarcos group from University of Castilla-La Mancha, Spain; and GES from Catholoc University of Brasilia, Brazil) wanting to collaborate on the definition of an ontology for software maintenance. Each research group had already published an ontology on software maintenance separately Ruiz *et al.* [22] and Días *et al.* [4] as a support for their respective ongoing research, but we perceived that each ontology bore the mark of its maker. Our goal in merging the ontologies was to obtain a more general maintenance ontology. Although we work under very strong restrictions, we also perceived positive factors that suggested that the work could be done.

What separates us:

- The Atlantic ocean (geographical distance);
- Five hours (different time zones);
- Two languages (Spanish and Portuguese although very close each other do not allow the easy discussion of such complex issues raised by an ontology merging).

The positive points:

- The domain, software maintenance, is relatively well defined.
- The researchers are all domain experts to some degree.
- Both ontologies are based on the same sources, the main ones being (see also Figure 1) an ontology for software maintenance [14] and another ontology for the software process [6]. Also used as a source by [14]. There are also a number of other minor sources in common such as international standards, significant publications, etc.

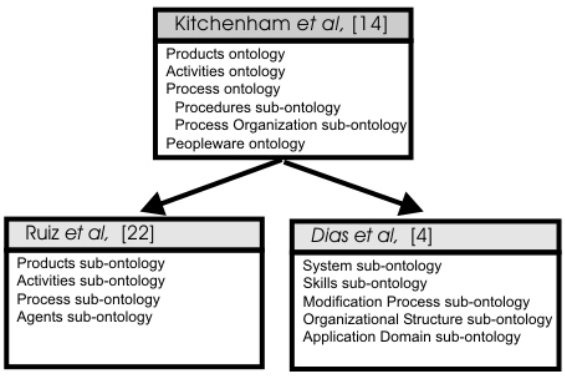


Figure 1. Schematic representation of three software maintenance ontologies

Before going on describing difficulties found and the process that we followed to merge the ontologies in more detail, we will describe the two source ontologies.

3.1. The two software maintenance ontologies

Table 1 and 3 characterize the two ontologies. Table 2 and 4 list some of the references used to build the two ontologies, including the common references highlighted in gray.

Table 1. Details of Ruiz *et al.*' ontology

Concept	Value
Domain	Management of Software Maintenance Projects
Author	Alarcos Research Group (UCLM)
	Ontology to enable information to be interchanged among engineers, managers and users fo maintenance projects
Level of formality	Semi-formal (REFSENO and UML)
Scope	List of concepts: This is classified(for reasons of clarity) into partial ontologies and subontologies: - Maintenance Ontology Products Subontology Activities Subontology Process Organization Subontology Procedures Requests Management Problems Agents Subontology - Workflow Ontology - Measurement Ontology
Source of knowledge	See table 2

Table 2. Sources of Knowledge used Ruiz *et al.*'s ontology

Informal ontology for SMP proposed by Kitchenham *et al.* in [14]
Conceptual model for corrective maintenance by Kajko-Mattson in [12,13]
Ontology for the software development process proposed by Falbo *et al.* In [6]
Conceptual model for software process and software measurement proposed by [1]
Documents which define the MANTIS processes system:
- Model of ISO 12207 life cycle
- Process reference model ISO15504-2 [9]
- ISO 14764 about SMP model [10]
- Model of activities and tasks of the MANTEMA methodology [20]

Ruiz *et al.*'s [22] ontology was focused on the concepts related to software maintenance projects from a static and dynamic point of view. Because of this the ontology also considers workflow and measurement issues. However, in this paper we only focus on the maintenance ontology from a static point of view, since these two issues were perceived as a specificity of Ruiz *et al.*'s ontology that Dias *et al.* did not consider in their work.

Dias et *al.*'s ontology was developed to describe the knowledge used in software maintenance. Therefore, the two ontologies, although focusing on software maintenance, have different goals, scope, organization (sub-ontologies). Note that Kitchenham *et al.*'s ontology, used as a source in both cases considered here, also has a different focus since it is aimed at classifying research in software maintenance.

Both source ontologies were modeled with UML. [7] state that UML may be used as a technique for modeling ontologies since it is easy to understand and use for people outside the AI community. Moreover, there is a standard graphical representation for UML models, and many CASE tools are available to manipulate these representations.

Table 3. Details of *Dias et al.' ontology*

Concept	Value
Domain	Practice of Software Maintenance Projects
Author	GES Research Group (Catholic University of Brasilia)
Purpose	Ontology to identify and organize all the knowledge needed when performing maintenance.
Level of formality	Formal (UML, dictionary of concepts, definition of restrictions in first order logic)
Scope	List of concepts: There are five subontologies: System Subontology Maintenance Process Subontology Computer Science Skills Subontology Organization Subontology Application Domain Subontology
Source of knowledge	More than 30 references, see Table 4 for principle sources

Table 4. Sources of Knowledge used Dias *et al.*'s ontology

Informal ontology for SMP proposed by [14]
Conceptual model for corrective maintenance by [13]
Ontology for the software development process proposed by [6]
Book on software Maintenance [19]
Books on Software Engineering [18, 21]
Standard on Software Maintenance IEEE-1219 [8], ISO 12207, and ISO 14764

3.2. The merging constraints

The work we undertook was to merge these two existing software maintenance ontologies into a new one. Contrary to the ideas proposed by [16] we did not adopt one "stable" (preferred) ontology on which to map the other. In this merge, the two source ontologies have exactly the same "weight". One reason for this is that both source ontologies are approximately of the same age and were too recent at the time we started the merging to have been used in other works. Therefore "changing" one or the other would have exactly the same (small) impact.

We started the work with each group studying the other's ontology. In this way we were able to get a better idea of what difficulties we would have, and what differences existed between this specific work and the merging process proposals found in literature which were of a more theoretical nature (meaning that the goal of these other works was not always to actually merge ontologies, but to propose processes or tools that would help in merging):

- The first conclusion was that the structure of the ontologies, although different overall had some commonalities: description (sub-ontology) of the software system maintained, description of the maintenance process, description of the organizational structure.
- A brief look at the concepts in the two ontologies showed that very few of them have similar names [16]: 11 concepts, which is about 15% of Ruiz *et al.*, and 10% of Dias *et al.*
- As already highlighted, an important constraint is that the two teams are geographically separated. This in itself ruled out most of the proposed merging tools which assume that all work is done at a "central" location.
- One ontology is described in a semi-formal way (see Tables 1 and 3) and the other more formally, but both use UML for graphical representation and textual description of the concepts (dictionary of concept).
- Each group has a biased understanding of the source material. Concretely, each group understands its own ontology and the rational for its design well, and the other ontology much less. There was no central authority which would have a balanced view of the sources ontologies.

We felt, and continue to believe, that the worst difficulty was the geographical dispersion. Merging two ontologies is very much like designing a new one[1]. It is essentially an exchange activity where experts confront their views to reach a common understanding. Doing so at distance proved a major difficulty.

3.3. The merging process

Because we were not able to exchange ideas quickly, we felt the necessity to have a well defined process to follow. As software engineers (primarily), we based this process upon some well known principles in our field, mainly from the USDP (Unified Software Development Process) [11], also known as RUP – Rational Unified Process). The USDP is a process for developing software, an activity that bears some resemblance to our situation:

- There is a highly conceptual initial part to software development (domain modeling) that is very similar to ontology modeling.
- Software development projects nowadays require the participation of many different people often in different places and speaking different languages (off-shore development).
- UML is the representation language of choice.

The positive practices that we wanted to incorporate from the USDP are:

- Iterative and incremental process: don't try to do everything at once but slowly (and iteratively) work toward the solution.
- Manage the project risks so that they don't suddenly jeopardize the project.

In the USDP, the iterative and incremental approach implies that all the functionalities are not implemented at once but spread out over various iterations, and that each functionality will itself be typically developed in several iterations (first the core functionality, then its alternative scenario). This is different from the iterative process proposed by [16], because in PROMPT, the iterations simply repeat the same activity whereas in USDP, each iteration is a small (sequential) process itself, including all activities from the most abstract (requirement elicitation, requirement analysis) to the most detailed (implementation and testing). In the case of ontology merging, we do not have the same abstraction span. However, one may still have different "activities" such as considering the subontologies, the concepts, the associations between them and finally the restrictions (for formal ontologies).

In the USDP, the iterations are a way of dealing with risks in software development: by developing the riskiest functionalities (only them and only their core feature) in the first iterations, one can evaluate more rapidly if one is able to implement them as planned. This allows better control over the whole project.

We attempted to apply these principles to the merging of ontologies by following the idea of an initial **core** (similar to the core functionalities of the USDP and their core scenario) that we could progressively expand to a complete ontology. This idea was applied on two different levels. First, at the ontology level, we started with a "core" sub-ontology which happened to be common to both ontologies to be merged (we will come back to this later). When we were satisfied that we would be able to merge this sub-ontology, even if the merging was not yet completed, we started to look for the next sub-ontology.

We applied a similar process at the level of concepts. When working on a sub-ontology, we focused first on a core concept (or a small group of core concepts) that we then expanded with related concepts, enlarging the scope to the point that we were satisfied that the sub-ontology considered was completely described. Thus in one iteration, we might be dealing with the core concepts of one sub-ontology

[1] Noy and Musen in [17] suggest that merging is actually a sub-case of designing a new ontology.

while still resolving some pending issues (of relatively little importance) of another sub-ontology.

The associations between the concepts proved to be easy to deal with once the concepts themselves had been agreed upon. Clearly, changing the meaning of concepts or adding new concepts had an impact on the associations, but this is normal and expected. Still we feel that focusing on the concepts first allowed to minimize the amount of re-work.

Finally in our case the restrictions are not an issue as one of the source ontology is semi-formal (therefore we do not need to merge restrictions).

3.4. Merging example: applying the iterative process

To better illustrate how we conducted the merging, we will now describe in some detail how we applied our process. Due to the geographical dispersion of the ontology designers, we adopted the following procedure: First, we agreed on a core sub-ontology to work on, then one team started working on the merging of this portion of the ontology. This team sent a proposal by e-mail to the other team which analyzed it, commented on it (accepting and/or counter arguing) and sent back its comments. This corresponds to one iteration where, as explained previously, we would work at various levels of detail (concept and/or associations). The proposal went back and forth between the two teams, until only minor issues remained pending. Then we started to look for the core concept of a second sub-ontology while the pending issues for the first were resolved ``in the background". We did this for each of the three sub-ontologies that we required.

The first sub-ontology we considered was the system ontology. When one talks about software maintenance, it seems clear to us that the software system to be maintained should be considered. Indeed, this was the only common sub-ontology that the two source ontologies had (in one ontology it was called Product sub-ontology instead of System sub-ontology) (see Figure 1). Although the two source ontologies agree on this, their respective software system sub-ontologies present significant differences: their names are different and they are at different levels of abstraction. In Ruiz *et al*. [22] the Product sub-ontology has only three concepts whereas there are 27 in Dias *et al.'s* [4] System sub-ontology. The two source sub-ontologies are presented in Figures 2 and 3.

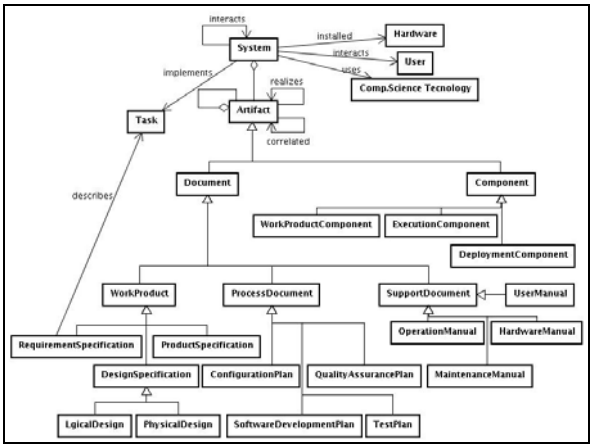


Figure 2. System sub-ontology [4]

To resolve this conflict, we applied the principle of finding the core concepts of this sub-ontology and expanded this core to the entire sub-ontology. Again these core concepts were what the two sub-ontologies had in common. This part proved to be similar to the initial step of the SMART algorithm which recommends the creation of a list of the concepts considered in each ontology [16], looking for concepts with identical names or with linguistically similar names.

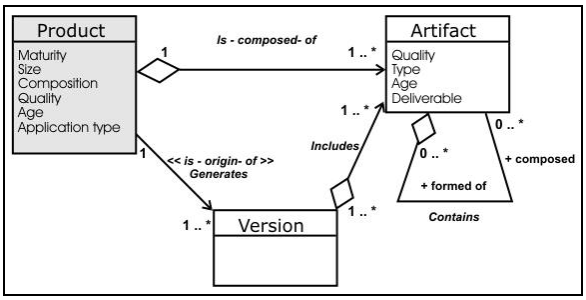


Figure 3. Products sub-ontology [22]

At this point we realized that, an automated tool would have had difficulties in identifying how to map the sub-ontologies as they used two different names (system and product), and their core concepts used different names too (again system and product).

From the three concepts in Ruiz *et al.* Product sub-ontology (Figure 3), two were found in Dias *et al.* System sub-ontology (Figure 2). These are the "software system" maintained and its "artifacts" (a software system is composed of artifacts). It seems clear to us that these two concepts are indeed core concepts of a System sub-ontology and could be used as a base to cultivate the entire sub-ontology. For this reason, we applied the "merge-class" operator, as described in [16]: the two "artifact" concepts were

merged in the new sub-ontology, and, the "system" and "product" concepts were merged as a "software product" concept. From this base, we expanded our initial sub-ontology. We decided to include the "version" concept present in one sub-ontology (Figure 3) and not the other (Figure 2), and the rest consisted of deciding what concepts from the System sub-ontology (Figure 2) would reach the merged sub-ontology (less important issues).

This is another point where an automated tool would have been of little use since the commonalities between the two sub-ontologies were very few (two concepts) and most of the work was discussed between the two groups to decide at what level of abstraction we wanted to work and what concepts of the System sub-ontology would be rejected or merged in the final sub-ontology.

The resulting sub-ontology is presented in Figure 4. We needed only one iteration (i.e. one round-trip: proposal from one group and answer/comment from the other) to agree on the core concepts of this first sub-ontology. The remaining (minor) issues were closed in another iteration (focused on the second sub-ontology).

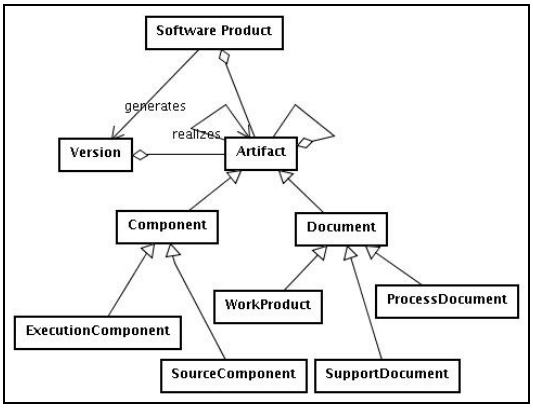


Figure 4. Merged Software Product sub-ontology

After having solved the main problems of the first sub-ontology, we continued. For the other sub-ontologies, we did not have a one to one correspondence between the two sources (one source sub-ontology would map to two, or more, in the other source), but having already defined a core sub-ontology and its concepts helped in merging the rest, because we had a common base to work from.

The next sub-ontology we considered was that of maintenance activity. This seemed to be the next logical sub-ontology to consider after that of system, first because it is also central to the idea of software maintenance, second because it is closely related to the system sub-ontology, and third because the two source ontologies had a sub-ontology relating to the maintenance process. This time, the two source ontologies had more differences since Dias *et al.* consider only one such sub-ontology whereas Ruiz *et al*. have two (see Figure 1). The idea of working iteratively in this case is important because it helps focus on a smaller part of the whole ontology. This is where the process allows better control on the whole project.

Existing tools are deficient in this regard since they appear to consider either the entire source ontologies to be merged (losing the focus we have just described) or would work on two given sub-ontologies (as in our first iteration), thus loosing some information since in this case the mapping is from two sub-ontologies (Ruiz *et al.*) to one (Dias *et al.*).

To merge the Process sub-ontologies, we again started from core concepts which we defined to be the "maintenance process" and its "activities". Interestingly, the very activity of looking for core concepts showed that if both source ontologies defined the "activity" concept neither had thought above the "process" concept. We agreed that this was an important concept to add. From these two concepts, we progressively added related concepts (for example a decomposition of the activities). This was more difficult than with the first sub-ontologies as we had to select concepts coming from both source ontologies, or either of them, or reconcile differing views on concepts. Things were even more difficult as we found that some interesting concepts were actually members of a fourth sub-ontology (e.g. the technology concept in Dias *et al.*'s Skills sub-ontology). We needed two iterations to settle the core of this sub-ontology, and one more for the minor issues.

The final sub-ontology to be merged is that of Organization which includes concepts on the roles needed to perform the activities, positions in the organization, etc. It is similar to the second one in that it consists of merging two sub-ontologies from one source with one sub-ontology from the other. It is also made more difficult by the fact that some concepts of these sub-ontologies have already been merged in the Process sub-ontology. Again, two iterations were necessary to agree on the core issues. Finally one more iteration was necessary to complete the work and solve the minor issues. In total, we needed six iterations.

4. Lessons learned

We have drawn the following conclusions from this merging experiment and how we dealt with the difficulties identified in Section 3.2:

- Core elements: As one would expect, in most cases the core elements (either sub-ontologies or concepts) we identified were the things the two source ontologies had in common. This is actually an approach adopted by most merging technologies. However, we found at least one example of a core concept (Process sub-ontology) that was not present in either source ontologies.
- Length: Some sub-ontologies required two iterations before we reached an agreement on the core issues and one more for the minor ones. Each iteration in this model corresponds to the analysis of a sub-ontology by one team, and the analysis of the resulting proposal by the other team (round-trip). Each of these analyses by one team could require weeks depending on the work load at that particular time. Typically, one month or more could pass between one group's proposal and the answer to this proposal. This was inevitable given the communication medium we chose (e-mail) and the geographical dispersion.
- Rework: The long intervals between each participation in the merging ultimately increased our work load as the first thing a team would have to do when commenting on a proposal would be to re-analyze the entire process to remind themselves of what had already been discussed, what had been proposed (and why) and what argument had been exchanged. In short, to reconstruct the entire discussion up to that point. A tool to help record and then reconstruct the discussion would be a great help in this sense.
- Communication channel: An ontology captures consensual knowledge in a domain. Reaching this consensus implies sharing ideas, confronting opinions, arguments and counterarguments. This is an activity that requires (a lot of) communication between the participants. Geographical dispersion is a huge obstacle to this communication. Because our objective was to actually merge the ontologies, we have not had time to research and develop a methodology that would alleviate this communication problem. Our method was a simple transposition to e-mail of what could have happened if we had been able to discuss the problem "eye-to-eye". This is definitely not enough, although it does offer some advantages as will soon be discussed.
- Iterative progress: Because the merging spanned a long time frame, it was important to have a structuring framework for discussion. One does not conduct a slow, lengthy, e-mail discussion in the same way as one would carry out an "eye-to-eye" exchange. It was important to have a clear sense of what we were doing at any given point, where we were going and how much was still needed to get there. This is one of the benefits of the iterative approach we used.
- Incremental progress: Usually, as one team was working on a piece of the ontology, the other would simply wait ("idly") for the next round of discussion. We did not concurrently start discussions on various sub-ontologies. This would not fit the incremental approach we chose. It is not clear whether this was a good decision or not. However the iterative and incremental approach allowed us to deal with the risk of making a wrong decision at one point that would imply re-working an entire sub-ontology. It is impossible from our limited experience to say if working concurrently on two sub-ontologies would not have increased such a risk.
- Process formality: Although we presented rather strict definitions of our process, its iterations and how they happened, in reality things are not so clear cut. For example we defined one iteration as a round trip of comments between the two groups. However, the real unit of activity was one analysis by a group ("half an iteration"). For example, a group would start a new iteration (discussion of the core issue for a sub-ontology) and at the same time (in the same e-mail) it would close the iteration of the preceding sub-ontology.
- Asynchronous communication: E-mail communication has the well known advantage of being asynchronous. One does not need to set appointments or wait for others. Given that the two teams are in different time zones this was a very good thing and proved useful.
- Historical record: E-mail communication and written communication in general also offers the advantage of being easily archived. This is important when one needs to go back to past decisions and remember how they were arrived at.

5. Conclusions

Merging methodologies is useful to guide the merging activity and to carry it out in a systematic and ordered way. Some automatic tools have been

proposed with the goal of making that activity easier. However, the merging process is very similar to developing a new ontology [7] since it is necessary to understand the source ontologies clearly, to decide the level of granularity of the final ontology and to make a lot of design decisions.

In this paper we have reported our experience in merging two ontologies on software maintenance in real life conditions. These conditions included: geographical dispersion of the participants, semi-formal definition of one ontology, two ontologies which were organized differently with few concepts clearly in common. In these conditions, we have found the use of automated merging tools to be of little value as most of the work consisted of discussion between experts to define what concepts to keep from either source ontology, what the exact definition of some concepts was, or at what level of granularity we wanted to work.

To carry out the merging, we defined an iterative and incremental process where the ontologies are merged iteratively by sub-ontologies and each iteration consists of an incremental approach from some core concepts to the entire sub-ontology.

We closed the paper with a discussion of our experience, highlighting some benefits and drawbacks of our approach.

6. References

[1] Becker-Kornstaedt, U., Webby, R.: A Comprehensive Schema Integrating Software Process Modelling and Software Measurement. . Fraunhofer IESE-Report Nº 047.99 http://www.iese.fhg.de/Publications/Iese_reports/ Fraunhofer IESE., (1999).

[2] Chalupsky, H.: OntoMorph: A Translation System for Symbolic Knowledge. KR'00, USA (2002).

[3] Deridder, D.: A Concept-Oriented Approach to Support Software Maintenance and Reuse Activities. Workshop on Knowledge-Based Object-Oriented Software Engineering at 16th European Conference on Object-Oriented Programming (ECOOP 2002), Málaga Spain (2002).

[4] Dias, M.G., Anquetil, N., et al.: Organizing the Knowledge Used in Software Maintenance. Journal of Universal Computer Science, Vol. 9 (2003) 641-658.

[5] Doan, A., Madhavan, J., et al.: Learning to Map Between Ontologies on the Semantic Web. Eleventh Intenational WWW Conference, Hawaii USA (2002).

[6] Falbo, R.A., Menezes, C.S., et al.: Using Ontologies to Improve Knowledge Integration in Software Engineering Environments. 4th International Conference on Information Systems Analysis and Synthesis (ISAS'98), Oraldo Florida (1998).

[7]Gómez-Pérez, A., Fernández-López, M., et al.: Ontological Engineering. (2004).

[8] IEEE: 1219 - Standard for Software Maintenance. IEEE - Institute of Electrical and Electronics Engineers (1998).

[9] ISO/IEC: 15504-2: Information Technology - Software Process Assessment - Part 2: A Reference Model for Processes and Process Capability. (1998).

[10]ISO/IEC: FDIS 14764: Software Engineering Maintenance (draft), Dec-1998. (1998).

[11]Jacobson, I., Booch, G., Rumbaugh, J.: The Unified Software Development Process. Addison-Wesley (1999)

[12]Kajko-Mattsson, M.: Common Concept Apparatus within Corrective Software Maintenance. IEEE International Conference on Software Maintenance (ICSM'99), Oxford UK (1999).

[13]Kajko-Mattsson, M.: Towards a Business Maintenance Model. IEEE International Conference on Software Maintenance (ICSM), Florence Italy (2001).

[14]Kitchenham, B.A., Travassos, G. H., et al.: Towards an Ontology of Software Maintenance. Journal of Software Maintenance: Research and Practice, Vol.11 (1999) 365-389

[15]McGuinness, D.L., Fikes, R., et al.: An Environment for Merging and Testing Large Ontologies. (2000)

[16]Noy, N., Musen, M.: An Algorithm for Merging and Aligning Ontologies: Automation and Tool Support. Workshop on Ontology Management, WS-99-13. AAAI Press, (1999)

[17]Noy, N., Musen, M.: PROMPT: Algotithm and Tools for Automated Ontology Merging and Alignment. Workshop on Ontologies and Information Sharing, Seattle Washington (2000).

[18]Pfleeger, S.L.: Software Engineering: Theory and Practice. Prentice-Hall (2001).

[19]Pigoski, T.M.: Practical Software Maintenance: Best Practice for Managing Your Investment. Jhon Wiley & Sons, New York USA (1997).

[20]Polo, M., Piattini, M., et al.: MANTEMA: A Complete Rigorous Methodology for Supporting Maintenance Based on the ISO/IEC 12207 Standard. Third Euromicro Conference on Software Maintenance and Reengineering (CSMR'99). IEEE Computer Society, Amsterdam Netherlands (1999).

[21]Pressman, R.S.: Software Engineering: A Practitioner's Approach, 5th edition. (2001).

[22]Ruiz, F., Vizcaíno, A., et al.: An Ontology for the Management of Software Maintenance Projects. International Journal of Software Engineering and Knowledge Engineering, Vol.14 (2004) 323-349.

[23]Steve, G., Gangemi, A., et al.: Integrating Medical Terminologies with ONIONS Methodology. Information Modeling and Knowledge Bases VIII. IOS Press (1998).

[24]Stumme, G., Meadche, A: FCA-MERGE: Bottom-Up Merging of Ontologies. Seventeenth International Joint Conference on Artificial Intelligence (IJCAI 2001). Morgan Kaufmann Publishers, Seattle, Washington (2001).

Hacia una Metodología Orientada al Conocimiento para la Educción de Requisitos en Ingeniería del Software

Alejandro Hossian, Enrique Sierra, Paola Britos, María Ochoa, Ramón García Martínez
Departamento Electrotecnia. Facultad de Ingeniería. Universidad Nacional del Comahue
Centro de Ingeniería del Software e Ingeniería del Conocimiento. Escuela de Postgrado. ITBA.
Laboratorio de Sistemas Inteligentes. Facultad de Ingeniería. Universidad de Buenos Aires.
hossi@ciudad.com.ar, esierra@uncoma.edu.a, {pbritos, maochoa, rgm}@itba.edu.ar

Abstract

Software Engineering entails an important phase which is the process of eliciting software requirements. In Software Engineering, eliciting techniques are not completely systematized, which makes the eliciting process difficult to deal with. In this process, the software engineer interacts with the mental model that the client has of reality and of the problem for which the software solution is being found. This is the reason for which the contribution of Knowledge Engineering is considered in this paper. Knowledge embedded in the client mental model allow for the construction of scenarios, which are representations of the reality that the client has in his or her mind. Since they are related, scenarios reflect the rate of the eliciting process. The contribution of multiple perspectives from different users and the incorporation of the characteristics of the software product to be developed lead to the construction of The Unified Map of Objective Scenarios. This is the final document contributed by the methodological approach contained in this paper for the representation of software solution requirements.

1. Identificación del Problema

Uno de los principales inconvenientes que se le presenta a la comunidad de desarrollo de software está relacionado con todo lo que se refiere a la gestión del proceso de requisitos [1]. En un sentido más amplio, se puede afirmar que los requisitos ya no son un problema exclusivo de la industria del software, y tampoco se puede afirmar que sean un problema exclusivo de los Sistemas de Información tradicionales. La cuestión de los requisitos puede ser abordada desde el enfoque más clásico, es decir, desde el *"Análisis de Requisitos"*, esto es como una actividad conducente a identificar los conceptos relevantes del problema para poder arribar luego a una solución software al mismo. O también dicha cuestión puede ser enfocada a través de la óptica de la *"Ingeniería de Requisitos"*, es decir a través de un proceso iterativo y cooperativo de análisis del problema que sea monitoreado y chequeado en forma permanente, de manera de poder custodiar eficientemente la calidad de la información obtenida.

A partir del enfoque clásico, es decir, desde el *"Análisis de Requisitos"*, se puede asumir que las tareas a llevar a cabo dentro de la actividad de análisis son: educción, modelización y validación.

La *educción* tiene como objetivo adquirir el conocimiento del dominio de los clientes y/o usuarios, de tal forma que sea posible identificar los conceptos, relaciones y funciones más relevantes. Por consiguiente, esta tarea tiene características típicas de investigación y es difícilmente formalizable, dada la gran variedad de conocimientos dependientes del dominio que puede ser necesario adquirir. En Ingeniería del Software, existen una serie de técnicas básicas para realizar esta tarea, como son las entrevistas (ya sean de carácter abierto o estructurado), los cuestionarios o el análisis de documentos, así como también algunas más elaboradas como el prototipado [2] y el JAD (Joint Application Development) [2]. De todas formas, el área de la informática que más ha tratado con la educción es la Ingeniería del Conocimiento, en la cual, además de las ya citadas, se han definido técnicas muy elaboradas como el análisis de protocolos [3] o el emparrillado [3].

La *modelización* consiste en representar, mediante la utilización de "Modelos Conceptuales", los conocimientos adquiridos en la tarea anterior. Los modelos conceptuales son mecanismos de representación, que permiten modelar los

conocimientos adquiridos durante la educción, con el fin de facilitar su comprensión y permitir su comunicación entre todos los participantes en la actividad de análisis (clientes, usuarios y analistas).

La *validación* consiste en verificar la exactitud de los conocimientos adquiridos, ya que no existe ninguna razón para que el analista sea inefable en su actuación.

La tarea de educción de requisitos presenta serias dificultades debido a que las técnicas que proporciona la Ingeniería del Software en este sentido no son lo suficientemente completas como para que la captura de requisitos del software sea la adecuada [4]. Sobre todo, cuando la complejidad de los conocimientos de un dominio de clientes y/o usuarios son de difícil comprensión. Es necesario ser consciente de que una educción pobre de los requisitos conlleva inexorablemente a un pobre modelado de los mismos, y por lo tanto, a un software de baja calidad [5].

2. Un abordaje para la solución desde la Ingeniería del Conocimiento

En este contexto, el problema al que se le pretende dar cobertura en este artículo, es el de asistir en la actividad de *Educción de Requisitos* en Ingeniería de Software (IS) con técnicas inspiradas básicamente en la *Adquisición de Conocimientos* de la Ingeniería de Conocimiento (IC). La educción de requisitos se refiere a la captura y descubrimiento de los requisitos. Entre los principales problemas que pueden entorpecer la tarea de educción de requisitos se cuentan los siguientes: los usuarios no pueden/saben describir muchas veces sus tareas, mucha información importante no llega a verbalizarse, la educción se afronta como un proceso pasivo, cuando debería ser un proceso cooperativo La asistencia a la actividad de Educción de Requisitos en IS desde la Adquisición de Conocimientos en IC, se ilustra en la Figura 1.

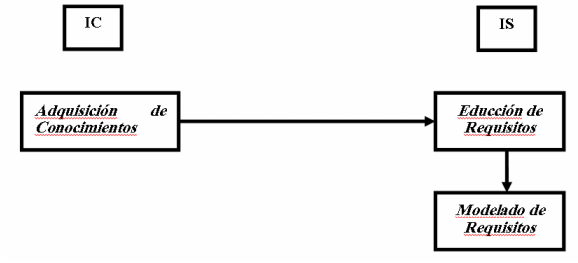


Figura 1. Asistencia de la IC a la IS

La idea directriz de este artículo es proporcionar un marco metodológico a esta asistencia, que ayude a sistematizar el proceso de educción. Para llevar el trabajo a su faz operativa, se dispuso de una serie de ejemplos de sistemas de información y a partir de una actividad de educción asistida por las técnicas provenientes de la ingeniería del conocimiento, se propuso llegar a obtener un instrumento que contenga la especificación de los requisitos del problema cuya solución se desea informatizar. El análisis de trabajos relacionados condujo a consultar documentación específica sobre la temática [6].

3. Características del proceso de educción de requisitos del software

Está claro que el ingeniero de requisitos, en su trabajo de educción, debe capturar y modelar una realidad que enmarca una problemática cuya solución se pretende realizar a través de un producto software. Al ser la realidad un elemento intangible, ¿cómo es posible capturarlo o más aún, modelarlo? Dado que el ingeniero de requisitos no tiene en principio un acceso directo a la realidad a modelar y por lo tanto tampoco a la problemática a resolver que está inmersa en esa realidad, debe valerse de elementos que le proporcionen una descripción de dicha realidad y su problemática, como así también de las características de la solución concebida inicialmente para resolverla. Esta descripción del problema y su solución pretendida por un producto de software estarán contenidas en declaraciones textuales de clientes o usuarios. Valiéndose de estas declaraciones como forma de acceso indirecto a la realidad a relevar y la problemática a resolver, el ingeniero debe determinar los requerimientos que debe satisfacer el producto software que resuelva dicha problemática con alcances y prestaciones también contenidos en declaraciones de clientes o usuarios. En definitiva, el ingeniero de requisitos no tiene acceso a la realidad y su problemática, sino a un cliente o usuario que vive o ha vivido esa realidad con el problema inmerso en ella y por lo tanto posee un capital en experiencia de la misma que el ingeniero no tiene. Ahora, este capital es básicamente intelectual, es un constructo: el cliente o usuario de una organización ha tenido vivencias de la realidad y del problema a abordar que le han permitido crear en su mente *un modelo de la misma*. Siendo la construcción de un modelo mental o de una estructura cognitiva un proceso individual indexado por vivencias y experiencias netamente personales que se dan en determinados contextos, es probable que el modelo mental creado por el cliente no se ajuste del todo a la realidad y la problemática que se desea abordar. Es probable que este modelo posea incompletitudes, incongruencias y hasta

inconsistencias derivadas de diversos factores que pueden ir desde una falta de contacto por parte del cliente o usuario con determinados aspectos de la realidad que son relevantes a la solución de la problemática en cuestión, hasta formas de interpretación de la realidad que no se ajusten objetivamente a los hechos cuya dinámica es preciso comprender para resolver adecuadamente la problemática planteada. Ahora, ¿Cómo superar este componente altamente subjetivo del proceso de educción de requisitos, que está presente en el discurso del cliente, cuando el mismo proviene de una representación mental cuya construcción es puramente personal y subjetiva?

Una forma de superar la subjetividad sería recurriendo a una multiplicidad de visiones de una misma realidad y la problemática a resolver, lo que hace pensar contribuiría a generar una visión convergente y por lo tanto con un mayor grado de objetividad y aproximación a los hechos tal como efectivamente ocurren. Estas perspectivas diferentes podrían ser aportadas por diferentes clientes, o por un mismo cliente adecuadamente seleccionado que haya pasado por diversas experiencias con relación al problema a modelar. En todo caso, este abordaje bajo múltiples perspectivas le permitirá al ingeniero de requisitos enfocar el problema desde distintos ángulos, y por lo tanto esto contribuirá a enriquecer significativamente la caracterización de los escenarios que contribuyan a configurar una visión integral de la realidad y su problemática.

4. Proceso de educción del conocimiento de usuario

Dado que el cliente o usuario habrá creado en base a su experiencia un modelo mental de la realidad y la problemática a resolver mediante un producto software, es necesario que el ingeniero de requisitos inicie de algún modo el proceso de construcción de un prototipo de forma de representación que en cierta manera replique o refleje el modelo mental presente en la estructura cognitiva del cliente o usuario. Conforme a investigaciones en Psicología Cognitiva [7], este modelo mental poseerá integrados distintos tipos de conocimiento: factual o declarativo, procedural o de procedimientos y de imágenes [8]. Dado que estas formas de conocimiento estarán interconectadas en la estructura cognitiva del cliente conformando un modelo mental de la realidad y su problemática, será posible a través del universo de discurso del mismo identificar estas formas de conocimiento y representarlas adecuadamente. Si se entiende a la realidad como una sucesión de situaciones que se dan en un determinado contexto, la representación de la misma según el modelo mental del cliente o usuario, se podrá realizar mediante una secuencia de escenarios. Formalmente, el concepto de escenario tal como se lo aborda en el presente artículo se define del siguiente modo:

Descripción, textual o gráfica, de una situación determinada que se da en el ámbito de aplicación del producto software a desarrollar para abordar la solución a un problema del cliente o usuario. Esta descripción ha sido tomada del propio universo de discurso del cliente o usuario respecto de su problema y se caracteriza por la interacción y vinculación, en un contexto dado, de actores o entidades cuyas propiedades y acciones están definidas por valores de atributo adecuadamente instanciados.

5. Pautas iniciales para la construcción de Escenarios

En base a lo expresado precedentemente, puede afirmarse que el escenario es situacional, es decir, describe una situación que se da en un determinado contexto. El escenario está compuesto por actores o entidades, que básicamente constituyen conceptos a los que hay que identificar y caracterizar a partir del discurso del cliente o usuario. Por lo tanto, para la adecuada caracterización de un escenario será necesario:

a) Identificar una situación concreta, un determinado estado de situación en el marco del problema que describe el cliente.

b) Identificar el contexto en el que se da esa situación, con sus restricciones o límites espaciales y temporales bien definidos.

c) Identificar las entidades o actores que son relevantes en este estado de situación conforme a las características del problema y dentro del marco contextual vigente.

Para cada actor en el escenario, es necesario identificar:

a) aspectos que ayuden a caracterizar propiedades relevantes al problema a los que se denominará atributos.

b) aspectos que contribuyan a caracterizar el estado del actor en el escenario actual, lo cual quedará definido por los valores de los atributos (instanciación del actor)

c) vinculación de un actor con otros actores. Esta vinculación quedará adecuadamente caracterizada mediante relaciones.

d) acciones que el actor realice y que sean relevantes al problema. Estas acciones no afectarán a otros actores en el escenario modificando valores de atributos de los mismos. Sólo alterarán (eventualmente) atributos del mismo actor que las realiza. Estas acciones también podrán estar caracterizadas mediante atributos.

e) Interacciones que el actor realice y que sean relevantes al problema. Estas interacciones se diferencian de las acciones en que afectarán a otros actores en el escenario determinando o modificando valores de atributo de los mismos. Estas interacciones también podrán estar caracterizadas mediante atributos.

f) El conjunto de acciones + interacciones constituye lo que se denominará la actuación de un actor en un determinado escenario.

Dado que el escenario , conforme a su definición, podrá ser la descripción gráfica de una situación determinada tomada de la realidad, que luego facilite la comprensión del problema y la comunicación entre clientes e ingenieros de requisitos, la representación esquemática de un escenario puede ser la ilustrada en la Figura 2.

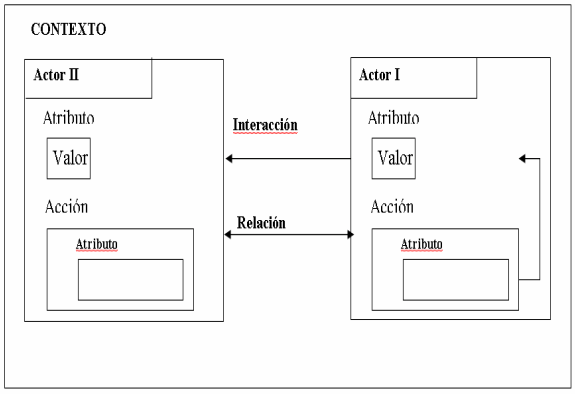


Figura 2. Representación esquemática de un escenario

6. Análisis cognitivo del discurso del cliente

El escenario adquiere el carácter de una forma de representación de tipo contextual que tratará de recrear la integración de los diferentes tipos de conocimiento, que en forma indexada por el contexto, han sido asimilados por el cliente en una situación determinada y estructurados en un modelo mental creado en base a sucesivas situaciones vividas en diversas experiencias ligadas a la realidad y a la problemática que se intenta capturar. Conforme a ello, y dado que una situación de la realidad se verá representada por un escenario, y teniendo en cuenta que la realidad puede ser vista como una sucesión de situaciones interconectadas, es por lo tanto razonable asumir que un conjunto de escenarios vinculados entre sí en una representación que se denominará *Mapa de Escenarios de Usuario*, constituirá una forma de reflejar el modelo mental internamente estructurado por el cliente en base a sus experiencias en la interacción con la realidad relativa al problema a resolver mediante una solución de software. Por lo tanto, el ingeniero de requisitos efectuará un *Análisis Cognitivo* de los sucesivos segmentos de discurso del cliente, a efectos de identificar los diferentes tipos de conocimiento embebidos en estos segmentos. Estas formas de conocimiento serán luego representadas en un cierto contexto en forma integrada, tratando de reconstruir, al menos en forma esquemática, aspectos relevantes de una situación vivida por el cliente y asimilada a su estructura cognitiva. Esta representación constituirá un escenario. La asociación de segmentos del discurso del cliente a situaciones concretas y su representación mediante escenarios constituye el punto de partida para comenzar a capturar la realidad asimilada por éste y su problemática. Para poder representar adecuadamente una situación mediante escenarios, es necesario efectuar un análisis cognitivo a los sucesivos segmentos de texto tomados a partir del discurso del cliente. Este análisis permitirá identificar [7]: *a) Conocimiento declarativo o factual:* se identificarán en el segmento de texto conceptos con sus atributos instanciados y acciones que los componen y caracterizan. También se identificarán las relaciones entre conceptos. *b) Conocimiento procedural o procedimental:* se identificarán en el segmento de texto procedimientos, métodos, formas de comportamiento, actividades, acciones que involucren la intervención de varios conceptos. *c) Conocimiento de imágenes:* se identificarán en el segmento de texto aseveraciones que involucren imágenes y la interpretación que les da el cliente o usuario. *d) Conocimiento del contexto:* se identificarán en el segmento de texto afirmaciones relativas al ámbito o entorno en el que transcurre la realidad descripta por el cliente. Una vez identificados los tipos de conocimiento presentes en los segmentos de texto, es posible comenzar a concebir la configuración de los diferentes escenarios asociados a dichos segmentos. Estos escenarios, adecuadamente interconectados,

constituirán una forma de representación del modelo mental que el usuario posee de la realidad y su problemática, a la cual se ha dado en denominar *Mapa de Escenarios del Usuario*.

7. Construcción de escenarios basada en el análisis cognitivo

A continuación se proveerán pautas para la construcción de los escenarios en base al conocimiento educido del universo de discurso del usuario. En este discurso se identificarán situaciones de la realidad, descriptas a través de frases que compondrán segmentos de texto. A estos segmentos de texto se le aplicará el análisis cognitivo a fin de identificar los tipos de conocimiento presentes en las frases del usuario, tal como fue explicitado en el apartado precedente. Una vez identificados estos tipos de conocimiento, se procederá a representarlos integrados en el constructo que se ha dado en denominar escenario. A tal efecto: a) El conocimiento contextual detallado por el usuario proveerá el marco espacial o ámbito en el que se desenvolverán los actores en el escenario. El contexto servirá a los efectos de suministrar un espacio de contención a la situación descripta en el escenario. b) El conocimiento declarativo o factual permitirá identificar conceptos, que se representarán como actores en el escenario. c) El conocimiento declarativo o factual permitirá caracterizar los actores del escenario, definiendo valores de atributo para los mismos. d) El conocimiento declarativo o factual permitirá definir las relaciones entre conceptos, que, de existir, serán representadas en el escenario como relaciones entre los actores del mismo. e) El conocimiento de imágenes permitirá asociar imágenes con actores, con sus atributos y eventuales acciones e interacciones. f) El conocimiento procedural o procedimental permitirá identificar acciones internas en los actores, las cuales podrán o no ser modificatorias de valores de atributo del actor. g) El conocimiento procedural o procedimental permitirá identificar interacciones entre actores, las cuales podrán modificar o no valores de atributo de los actores que interactúan. Por cuanto los escenarios de usuario quedarán definidos mediante los tipos de conocimiento embebidos en las declaraciones del mismo usuario cuando describe el problema a resolver mediante una aplicación de software. Sin embargo, estos escenarios no describen situaciones inconexas, sino que las mismas estarán interconectadas configurando el *Mapa de Escenarios de Usuario* que será una forma de representación que debe aproximarse lo más posible al modelo mental de la realidad y su problemática como ha sido construido por el usuario en su interacción con la misma. Ahora, surge la pregunta: si la realidad a educir del usuario puede ser vista como una sucesión o al menos, coexistencia de escenarios en una forma de representación integrada que se ha dado en denominar *Mapa de Escenarios*, ¿Cuál es el límite para delimitar cada escenario? ¿Adónde comienzan y terminan los segmentos de texto que delimitan a los escenarios? A continuación se dan pautas para delimitar la configuración de los escenarios a partir de la segmentación del universo de discurso del usuario:

a) *Cuando el usuario en su discurso indica un cambio de contexto*, es decir un cambio del marco espacial (o eventualmente temporal) en el que transcurre la situación que describe, entonces el ingeniero de requisitos debe asumir que el usuario está describiendo un nuevo escenario. b) Cuando el usuario en su discurso refleja un cambio de estado en los actores que componen un escenario dado, de tal modo que este cambio de estado provoca cambios en los valores de los atributos de los actores, se considera que los *actores con los nuevos valores de atributo pasarán a formar parte de un nuevo escenario*. c) Cuando el usuario en su discurso habla acerca de *eventos que modifican sustancialmente la composición del escenario actual*, como por ejemplo la adición de nuevos actores.

8. Pautas para la construcción del mapa de escenarios de usuario

La interconexión de escenarios marca una secuencia en la ocurrencia de las diversas situaciones contenidas en el discurso del usuario, denotando el ritmo del proceso de educción. Cuando en la descripción del problema suministrada por el cliente la ocurrencia de ciertos escenarios está condicionada a la ocurrencia previa de otro u otros, se identifica una relación de precedencia entre escenarios, lo cual hace que los mismos aparezcan en el Mapa de Escenarios conectados por una flecha con un sentido que va del escenario o escenarios iniciales o de partida al que le sucede. Existirán también situaciones cuya ocurrencia será identificada por el cliente en forma aislada de otras, pero que deben ser tenidas en cuenta a la hora de modelar la realidad y su problemática. Estas situaciones se representarán en el Mapa de Escenarios como elementos inconexos, dado que su ocurrencia no es activada por situaciones previas y la misma tampoco es necesaria para dar lugar a nuevas situaciones. Sin embargo, de algún modo, el Mapa de Escenarios indica una secuencia espacio-temporal que intenta reconstruir

el pensamiento existente en la mente del cliente. Por lo tanto, el Mapa de Escenarios será una herramienta que le permitirá al ingeniero de requisitos "desplazarse" por los diferentes caminos descriptivos de la realidad tal como la imagina el usuario, constituyéndose en una especie de "guía acerca de cómo el usuario tiene estructurados sus pensamientos concernientes a la realidad y el problema a abordar que está inmerso en ella". Está claro que el Mapa de Escenarios de Usuario al que finalmente arribe el ingeniero de requisitos habrá sido suficientemente "consensuado" con el usuario en una especie de proceso de negociación de carácter iterativo. Dicho proceso estará basado en una serie de interacciones con el cliente a fin de que éste otorgue su conformidad acerca de la representación de la realidad que el ingeniero le plantea.

9. Múltiples perspectivas en la visión de usuario: convergencia al mapa unificado

Está claro que la descripción de una realidad con el problema a solucionar mediante software, cuando es aportada por un único usuario resulta en general insuficiente para su adecuada interpretación y comprensión. Esto se debe a los aspectos subjetivos involucrados en la visión de la realidad que un único usuario incluye en su discurso del problema. Dado que el ingeniero de requisitos no tendrá, en general, un acceso directo a la realidad que debe relevar, debe recurrir al relato de las personas que sí tienen o han tenido un contacto con ella. La idea subyacente en el proceso de educción es que diferentes usuarios, en su posición de "observadores" de la realidad y su problemática aportarán diferentes Mapas de Escenarios, por los aspectos propios de su visión personal y porque distintos usuarios han vivido diferentes experiencias en contacto con dicha realidad. Es por ello que el ingeniero de requisitos debería, mediante una serie de interacciones con estos usuarios, arribar a un *Mapa de Escenarios Unificado* en el que converjan las distintas visiones de los usuarios consultados, y que los satisfaga en cuanto a la realidad que este Mapa intenta describir o comunicar. En el proceso de educción, el ingeniero de requisitos arribará a distintos Mapas de Usuario, conteniendo escenarios que pueden resultar incompletos, irrelevantes, inconsistentes o en algunos aspectos no coincidentes con los escenarios que surgen de la interacción con otros usuarios. En esta fase del proceso de educción, el ingeniero inicia un proceso de consultas a los usuarios involucrados a efectos de arribar a visiones convergentes de la realidad y el problema a resolver, documentando la misma en el *Mapa Unificado de Escenarios de Usuario.*

10. Escenarios centrados en objetivos

El documento referido al Mapa Unificado de Escenarios de Usuario constituye una representación que intenta aproximarse a una visión integral de la realidad tal como es percibida por los usuarios involucrados en el problema, cuyas descripciones contribuyeron a formalizar en el mencionado Mapa. La comprensión de la realidad y su problemática es esencial para el abordaje de la solución software [9], sobre todo en lo que respecta a las características que deberá poseer el producto que haga operativa dicha solución. En otros términos, es necesario reconocer la vinculación que existirá entre la realidad y su problemática y la solución software producida para resolver esta problemática. De algún modo, esta solución tomará aspectos de interés de la realidad-problema y los procesará en función de los requisitos manifestados por el usuario. En este sentido, el ingeniero de requerimientos deberá poder elaborar un documento, que conteniendo características similares a los escenarios de usuario, capture de los mismos información que sea relevante en función de las prestaciones requeridas para la solución software que se pretende desarrollar. Este documento, denominado *Mapa de Escenarios – Objetivo*, consistirá en un encadenamiento de representaciones que respetará la misma estructura que el Mapa de Escenarios Unificados de Usuario. Cada una de estas representaciones, a la que se denotará por *Escenario-Objetivo*, constará de dos bloques interconectados, identificados como Espacio-Problema y Espacio-Producto. El bloque identificado como Espacio-Problema, coincide en su representación con el correspondiente escenario en el Mapa Unificado de Escenarios de Usuario. Sin embargo, en el espacio-problema se identifican aquellos elementos del escenario de usuario que son requeridos por el producto o aplicación software en función de sus necesidades de procesamiento, en conformidad con los requerimientos de los usuarios. Esta identificación puede estar representada gráficamente por medio de un círculo que incluya a cada elemento del espacio-problema requerido por el producto para ser procesado. El bloque identificado como Espacio-Producto, contiene los distintos tipos de procesamiento, que basados en los elementos enmarcados (identificados con un círculo) en el espacio-problema, dan lugar a las funcionalidades requeridas por los clientes-usuarios. Una flecha

dirigida desde el elemento enmarcado a un cuadro indicativo del tipo de procesamiento a realizar, vincula los bloques representativos de los espacios problema y producto. De este modo, el procesamiento es visto como una funcionalidad que se aplica sobre una realidad ya modelada en los escenarios de usuario y sobre la cual se efectúan restricciones (al realizar la identificación de elementos en el espacio-problema) en términos de los requerimientos a los que debe dar respuesta la solución software a desarrollar. Esto puede observarse a modo de ejemplo para un escenario hipotético representado en la Figura 3.

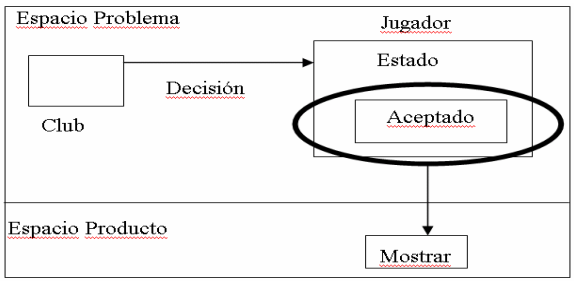


Figura 3. Representación de un Escenario – Objetivo

11. Síntesis de la propuesta metodológica

Las actividades descriptas previamente pueden relacionarse y resumirse en las siguientes etapas propias de una propuesta metodológica para la educción de requisitos del software que se ha dado en llamar Técnica de Educción Basada en Escenarios (TBE): 1) Recepcionar y documentar el discurso del cliente y sus necesidades de información. 2) Evaluación del discurso del cliente frase por frase 3) Identificación, en el discurso del cliente de escenarios y de transiciones entre los mismos. 4) Segmentación del discurso del cliente mediante la asociación de cuerpos de texto a escenarios 5) Análisis cognitivo del discurso del cliente, identificando los tipos de conocimiento embebidos en los cuerpos de texto. 6) Construcción de los escenarios en una secuencia que denote el ritmo de la educción 7) Refinamiento iterativo del escenario por interacción con el cliente y construcción del *Mapa de Escenarios de Usuario* 8) Interacción con distintos usuarios y construcción del *Mapa Unificado de Escenarios de Usuario*. 9) Identificación de las características de la solución software, con especial atención en los objetivos de información requeridos por los usuarios 10) Construcción del *Mapa Unificado de Escenarios Objetivo*. El marco metodológico descripto se ilustra en la Figura 4.

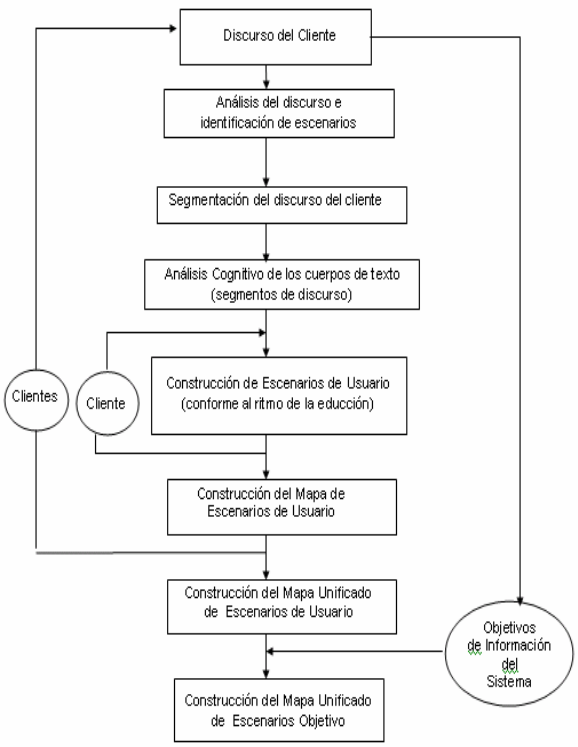


Figura 4. Síntesis del marco metodológico

12. Conclusiones

La utilización de la TBE en diferentes casos de desarrollo de software ha mostrado capacidad de la técnica para contribuir a sistematizar el proceso de educción de requisitos.

Al basarse en un análisis cognitivo del discurso del cliente, la técnica captura aspectos del modelo mental que el mismo tiene del problema, lo cual es esencial para una adecuada comprensión de sus características. La representación de este modelo mental mediante escenarios permite visualizar en forma gráfica el pensamiento del usuario, lo cual puede ser contrastado con éste en sucesivas instancias iterativas de refinamiento. Los escenarios muestran en general una dependencia para su ocurrencia, reflejando el ritmo de la educción. Esta representación temporal de una sucesión de situaciones encadenadas le permite al ingeniero de requisitos visualizar la realidad como es vista por el usuario en un mapa integrado por diferentes contextos altamente relacionados. La incorporación de múltiples perspectivas de usuario al mapa permite el ajuste de los escenarios conforme al aporte de diferentes visiones del problema. Finalmente, los objetivos de información del software a desarrollar permiten incorporar al modelo unificado, que

originalmente se había orientado al problema, las especificaciones propias del producto software. Asimismo, la TBE ha mostrado en los diferentes casos en que se la ha aplicado fortalezas para sistematizar, a partir de mecanismos concretos, el proceso de modelado relacionando el mapa unificado de escenarios objetivo con diferentes aspectos de la construcción del modelo del software, la segunda etapa propia del Análisis de Requisitos.

13. Referencias

[1] Demarco, T. 1979. *Structured Analysis and System Specification*, Yourdon Press.

[2] Davis, A. 1993. *Software Requirements: Objects, Functions and States*, Prentice Hall,

[3] Gomez, A, Juristo N., Pazos J., Montes C. 1997. *Ingeniería del Conocimiento* Editorial Ceura,

[4] Alford, M. 1977. *A Requirements Engineering Methodology for Real-Time Processing Requirements. IEEE Transactions on SoftwareEngineering*, SE-3(1).

[5] Yeh, R., Zave, P. 1980. *Specifiying Software Requirements*, Proc. of the IEEE, 68(9): 1077-1085.

[6] Juristo Juzgado, N. 1991. *Método de construcción del núcleo de una base de conocimientos a partir de un modelo de clasificación documental*. Tesis doctoral de la Facultad de Informática. Universidad Politécnica de Madrid.

[7] Manilow, A. 2005. *Teaching proportion word problems using a multiple linked-representational software design* – Columbia University doctoral dissertation (Subjective Area: Cognitive Psychology) –Advisor: John Black.

[8] Anderson, J. 2006. *Cognitive Psychology and its implications* – Watson Guptill Publications.

[9] Erdmann, M. and Studer, R. (1998). Use-Cases and Scenarios for Developing Knowledge-based Systems. In: *Proc. of the 15th IFIP World Computer Congress, WCC'98, Conference on Information Technologies and Knowledge Systems, IT&KNOWS'98* (J. Cuena, ed.), pp. 259-272.

Casos de (Re)Uso: Uma Abordagem para Reuso de Software Interativo Dirigida por Casos de Uso e Padrões Concretos de Interação

Augusto Abelin Moreira [1,2]
[1] *Companhia de Processamento de Dados do Estado do Rio Grande do Sul (PROCERGS)*
Caixa Postal 236 – 90010-340
Porto Alegre – RS – Brasil
+55 51 3210-3100
augusto@procergs.rs.gov.br

Marcelo Soares Pimenta [2]
[2] *Instituto de Informática*
Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91501-970
Porto Alegre – RS – Brasil
+55 51 3316 6814
{aamoreira ,mpimenta}@inf.ufrgs.br

Abstract

This paper aims to present an use case driven software reuse approach for interactive systems, integrating – by means of some aspects of use case life cycle (from modeling to implementation) - several well-known reuse concepts and techniques like use case patterns, interaction patterns and design patterns. The approach focuses on how to promote user interface reuse integrated to reuse of application-domain related software artifacts and by means of concrete interaction patterns (CIPs) usage. A CIP extends the usual interaction pattern documentation describing interactions and user interface behaviour through User Interface Description Language (UIDL), allowing reuse and implementation of an abstract user interface to different platforms and technologies.

Keywords: use case driven reuse, reified use case pattern, user interface reuse, software reuse, use case presentation, interaction pattern, concrete interaction pattern, User Interface Description Language, UIML.

Resumo

O objetivo deste artigo é apresentar uma abordagem de reuso de artefatos de software interativo dirigida por casos de uso, integrando – por meio de alguns aspectos do ciclo de vida de casos de uso (da modelagem à implementação) – vários conceitos e técnicas de reuso bem conhecidos como padrões de casos de uso, padrões de interação e padrões de projeto. Um dos focos da abordagem é como promover o reuso de interface com o usuário integrado ao reuso dos artefatos orientados ao domínio da aplicação através da especificação e utilização de modelos de reificação de padrões de interação, os quais denominamos: padrões concretos de interação (concrete interaction patterns – CIPs). Um padrão concreto de interação estende a documentação de um padrão de interação com a descrição das suas interações em uma linguagem de descrição de interface com o usuário (UIDL). Desta forma, explicitam-se os elementos de IU necessários para realizar a interação bem como os seus comportamentos, o que torna possível reusar e derivar implementações para as mais diversas plataformas e tecnologias.

Palavras-chaves: reuso dirigido por casos de uso, padrão de caso de uso reificado, reuso de interface com o usuário, reuso de software, apresentação de caso de uso, padrão de interação, padrão concreto de interação, Linguagem de Descrição de Interface com o Usuário, UIML.

1. Introdução

Embora o reuso seja uma das práticas mais indicadas da Engenharia de Software, a sua aplicação em IHC é ainda pouco difundida, talvez pela crença muito presente entre os designers de que reuso de soluções de outrem é um sinal da diminuição da sua criatividade. No entanto, o reuso em software é visto por muitos autores como uma das mais prováveis "balas de prata" para solucionar os problemas do desenvolvimento de sistemas [8].

Sua adoção na prática de design de interação por profissionais de IHC potencialmente permitiria uma

maior qualidade deste design (reuso de uma solução bem sucedida em teoria diminui a probabilidade de ocorrência de erros) e uma maior produtividade (reuso de elementos libera os designers para tratar problemas para os quais ainda não há solução) da equipe.

Mas a adoção de práticas de reuso não é implantada facilmente em uma organização. Construir artefatos reusáveis requer informações de identificação, extração, organização e representação de uma maneira que seja fácil de entender e manipular. Encontrar os artefatos reusáveis que possam ser (re)utilizados para o desenvolvimento de um novo sistema pode ser muitas vezes mais difícil e trabalhoso do que o desenvolvimento deste novo sistema. De fato, software para ser reusável e reusado deve ser projetado, documentado e implementado para este fim de reuso [25].

Embora a tecnologia disponível já habilite os designers a praticar reuso, isto não implica que o reuso ocorrerá. É preciso uma estratégia de reuso (com conjunto de conceitos e ferramentas associados) que propicie o reuso em diferentes níveis de abstração. Tal estratégia deve focar em reuso durante todo o ciclo de desenvolvimento, prover suporte à reutilização de artefatos (development with reuse) e à produção de artefatos reusáveis (development for reuse), ser facilmente integrável a outros métodos e técnicas de desenvolvimento utilizados e ser implementada por meio de ferramentas (também integráveis a ferramentas correntemente utilizadas) [20].

Desenvolver interfaces usando um processo de reuso possui basicamente 3 etapas: 1) realizar buscas em algum repositório de elementos reusáveis (assets), 2) selecionar os elementos mais adequados e 3) adaptá-los ao novo contexto específico de uso. Hoje, os assets de IHC mais comuns presentes em um repositório são os objetos de interação - num nível de implementação - e os padrões de interação (interaction patterns) - num nível mais abstrato.

Um padrão de interação captura conhecimentos comprovados de projeto de interfaces e é descrito em termos de um problema, um contexto e uma solução, no mesmo estilo dos padrões GoF [12]. Em particular, um padrão de interação deve estar focado em soluções que melhorem a usabilidade do sistema em uso [29].

O objetivo deste artigo é apresentar uma abordagem para reuso de interfaces com o usuário (IUs) através da especificação e utilização de modelos de reificação de padrões de casos de uso e padrões de interação. O artigo está estruturado da seguinte forma: a seção 2 apresenta uma revisão sobre os conceitos fundamentais dos padrões de interação e a sua aplicação no projeto de sistemas interativos e a seção 3 resume os principais conceitos envolvidos na descrição de IUs através de linguagens baseadas em XML. A seção 4 apresenta o conceito de padrões concretos de interação, os ilustra através de um exemplo e discute como podem ser utilizados em práticas de reuso. O projeto de IU em um processo de desenvolvimento dirigido por casos de uso e o papel dos padrões de casos de uso neste processo são discutidos nas seções 5 e 6, respectivamente. Na seção 7, apresentamos o fio condutor da nossa abordagem de reuso que é o padrão de caso de uso reificado. O processo de identificação de artefatos de IU reusáveis integrado a um processo de desenvolvimento OO é apresentado na seção 8. Finalmente, a seção 9 é a conclusão.

2. Reusando padrões de interação no projeto de interfaces com o usuário

Os primeiros textos relacionados especificamente ao desenvolvimento de IU associados a padrões de interação *(interaction patterns - IP)* começaram a surgir a partir de 1994 [18]. Entretanto, somente a partir dos trabalhos de Borchers [3, 4], Sutcliffe e Carroll [21] e Tidwell [23] é que se deu um incremento da aplicação de padrões na área de Design de Interação.

Os padrões raramente existem de forma isolada. As linguagens de padrões (*pattern languages*) reunem padrões que se relacionam e se complementam entre si, disponibilizando para o projetista de interfaces um acervo de idéias comprovadas de interação que podem ser aplicadas no seu projeto de forma consistente [24]. Entretanto, as linguagens de padrões ainda não são utilizadas de forma sistemática nos projetos de interface talvez por existir uma multiplicidade de linguagens (ver p.ex: 9, 13, 14, 23 e 28) muitas vezes incompatíveis entre si [10].

No estágio atual, os padrões de interação têm um potencial muito grande para serem úteis no reuso de projeto de interfaces. No entanto, para isto ocorrer de forma efetiva, algumas questões precisam ser endereçadas. Uma delas se refere à padronização do formato. Embora existam propostas de padronização do formato do padrão (entre elas [11] e [16]), seu objetivo é estruturar a documentação do padrão e não a descrição da IU que implementa a sua solução. Outra questão está relacionada à especificação das interfaces que implementam o padrão: não existem descrições precisas dos elementos de interação e como eles devem se comportar numa implementação da solução proposta pelo padrão de interação. De fato, pode haver diferentes implementações do mesmo padrão com

diferenças de comportamento entre elas. Para ilustrar isto, considere uma IU que implementa o padrão de interação "Parts Selector" da coleção de Van Welie [22] apresentada na Figura 1.

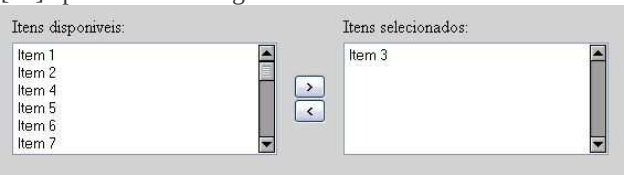


Figura 1. Uma IU do padrão de interação "Parts Selector"

A documentação deste padrão define claramente o trinômio problema-contexto-solução bem como apresenta exemplos de uso deste padrão. Entretanto, esta documentação não é suficiente para que um projetista de interface possa entender o funcionamento da interface e implementá-la de uma forma consistente com as demais implementações da mesma interface. Muitas questões ficam em aberto. Por exemplo: um item a ser adicionado na caixa de listagem "Itens selecionados" será inserido no final da lista ou no lugar que ocupará em uma lista ordenada? Os itens que são movimentados da caixa "Itens disponíveis" para a caixa "Itens selecionados" são removidos ou permanecem na caixa "Itens disponíveis"? Podem-se selecionar múltiplos itens ou somente um de cada vez? Como funciona a interface com o uso do teclado? Os botões de movimentação ficam habilitados quando não há item selecionado ou uma caixa de listagem fica vazia?

Neste artigo, propomos que a documentação do padrão de interação seja estendida por um conjunto de descrições das IUs que o implementam. Desta forma, as dúvidas de funcionamento da IU são dirimidas e - o que é mais importante - dá consistência às IUs pelo fato de poderem ser implementadas da mesma maneira.

Com a descrição das IUs, vislumbram-se oportunidades de reuso não só das (boas) idéias do padrão de interação, mas também de inúmeros artefatos que vão desde reuso das descrições de IUs nos projetos de interface até o reuso direto de implementações destas IUs.

3. Descrevendo as interfaces com o usuário

Uma IU pode ser descrita através de hierarquias de elementos de interação e como eles se comportam. Estas hierarquias podem ser representadas em vários níveis de abstração que vão desde descrições conceituais até as implementações da IU em uma determinada tecnologia.

Dentre as várias abordagens de descrição existentes, a nossa abordagem foi inspirada no modelo de desenvolvimento de IUs para aplicações interativas multi-contexto, o CRF - Cameleon Reference Framework [5]. O Cameleon Framework define modelos que suportam os vários níveis de descrições de IU (que no CRF são chamados de passos desenvolvimento), bem como os processos para a transformação de uma descrição de IU em outra mais concreta (processo de reificação); em outra mais abstrata (processo de abstração); ou em outra em um contexto de uso diferente num mesmo nível de abstração (processo de tradução). A Figura 2 ilustra os dois passos de desenvolvimento do CRF com os processos que serão usados na nossa abordagem.

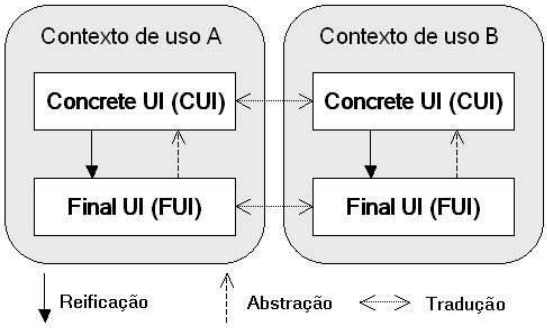


Figura 2. Os dois níveis mais "concretos" do Cameleon Framework.

Uma *Final UI (FUI)* é uma IU operacional, ou seja, qualquer IU executando em uma plataforma de computação específica. Os artefatos de uma FUI são os códigos-fonte ou os códigos-objeto que implementam a IU.

Uma *Concrete UI (CUI)* é uma representação abstrata de uma FUI de forma que seja independente de qualquer plataforma computacional ou de toolkit de desenvolvimento. Uma CUI consiste de uma decomposição hierárquica de CIOs de um dado contexto de uso. Um CIO *(Concrete Interaction Object)* é definido como qualquer entidade de IU que o usuário pode perceber (tais como texto, imagem e som) e/ou manipular (tais como botões, caixas de listagem ou menus) [27].

O artefato mais usado para descrever uma CUI é através de uma Linguagem de Descrição de IU (*User Interface Description Language - UIDL*). As UIDLs são linguagens baseadas em XML que permitem descrever as IUs de forma independente de tecnologia e de ambiente de desenvolvimento integrado (IDE). Atualmente, existem várias propostas de UIDL tais como: AUIML, UIML, UsiXML, XIML e XUL, para

citar algumas. Nós optamos por descrever as CUIs em UIML [26] por dois motivos básicos: porque é um dos padrões abertos de UIDL mais utilizados atualmente e porque tem grande flexibilidade para definir quaisquer conjuntos de elementos de interação (widgets) através dos *vocabulários*.

A Figura 3 apresenta uma CUI escrita em UIML para a interface apresentada na Figura 1.

```
<?xml version="1.0"?>
<!DOCTYPE uiml PUBLIC "-//Harmonia//DTD UIML 3.0a
Draft//EN" http://uiml.org/dtds/UIML3_0a.dtd>
<uiml>
<peers> <presentation
base="GenericJH_1.2_Harmonia_1.0"/></peers>
<interface>
 <structure>
  <part class="G:TopContainer" id="top">
   <style> ... </style>
   <part class="G:TextBox" id="textDisp">
    <style>
     <property name="g:textboxtype">textarea</property>
     <property name="g:text">Itens disponiveis:</property>
     <property name="g:editable">false</property>
    </style>
   </part>
   <part class="G:List" id="listaDisp"> ... </part>
   <part class="G:Button" id="botaoAdiciona" >
    <style>
     <property name="g:text">&gt;</property>
     <property name="g:buttontype">push</property>
     ...
    </style>
   </part>
   <part class="G:Button" id="botaoRemove" > ... </part>
   <part class="G:TextBox" id="textSelec"> ... </part>
   <part class="G:List" id="listaSelec"> ... </part>
  </part>
 </structure>
 <behavior> ... </behavior>
</interface>
</uiml>
```

Figura 3. Extrato de uma CUI para o padrão "Parts Selector"

4. Ligando padrões de interação a linguagens de descrição de interfaces com o usuário

Na prática, um padrão de interação precisa ser implementado utilizando algum conjunto de objetos de interação disponível em alguma plataforma e descrito em alguma linguagem. Obviamente, um mesmo padrão de interação pode ser mapeado para diferentes formas de interação (modalidades) e de implementação (tecnologias). A nossa proposta busca uma abordagem de reuso de IHC ao aglutinar os conceitos de padrão de interação com os de descrição de IUs no que chamamos de *padrão concreto de interação*.

Um padrão concreto de interação *(concrete interaction pattern – CIP)* é um conjunto de artefatos que descrevem e implementam uma IU de um padrão de interação. O CIP estende a documentação do padrão de interação com a agregação de modelos notacionais para a reificação das IUs. Estes modelos estão estruturados em um CIP conforme apresentado na Figura 4.

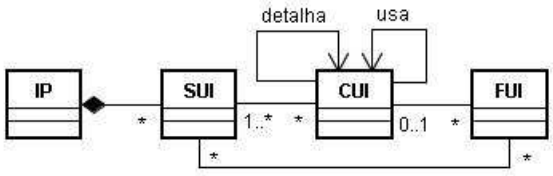


Figura 4. Estrutura de um padrão concreto de interação.

Um padrão de interação (IP) pode conter várias SUIs (Sketched UIs). As SUIs nada mais são do que um esboço (desenho, screenshot, etc) e uma descrição textual da IU que será descrita e/ou implementada pelos demais artefatos do CIP. O screenshot da Figura 1 ilustra uma SUI do padrão de interação "Parts Selector".

Cada SUI pode conter um conjunto de CUIs que a descrevam em diferentes níveis de detalhamento e para diferentes modalidades de interface (conjunto de widgets) e tecnologias de implementação (plataformas/linguagens). Uma CUI pode encapsular (usar) CUIs de outros CIPs. Com isso, além da prática de reuso que o encapsulamento proporciona, consegue-se descrever como vários padrões de interação funcionam de forma integrada num mesmo contexto de IU. Esta prática reforçaria o desenvolvimento de uma *linguagem de padrões* na coleção de padrões de interação sendo utilizada.

Finalmente, uma SUI pode estar associada a várias FUIs que a implementam em tecnologias, linguagens ou frameworks distintos.

Todos os artefatos que constituem o CIP são obtidos através de processos de reificação (abordagem top-down), abstração (abordagem botton-up) ou tradução descritos no CRF. A nossa abordagem não define se e em qual ordem devem ser construídos. Entretanto, os que são construídos num mesmo "caminho" de reificação/abstração devem ser compatíveis entre si.

Quanto mais alta a abstração de um CIP, maior a sua possibilidade de reuso. Para aumentar as possibilidades de reuso do CIP e facilitar os processos de abstração, reificação e tradução, sugere-se descrever uma SUI em 3 CUIs com diferentes níveis de detalhamento (CUI básica, intermediária e completa). A CUI básica contém apenas a discriminação dos elementos de interação, o layout e descrições textuais de

comportamentos relevantes da IU. A CUI completa contém uma especificação detalhada o suficiente para se gerar uma FUI a partir dela. Obviamente, a CUI intermediária, contém descrições no meio termo entre as duas.

Para ilustrar o processo de construção de um CIP, considere o diagrama da Figura 5 que apresenta a composição atual dos artefatos do CIP "Parts Selector". Algumas seqüências possíveis de construção destes artefatos seriam: 1-2-3-4-5-6, 2-4-5-6-3-1 e 1-6-2-5-4-3.

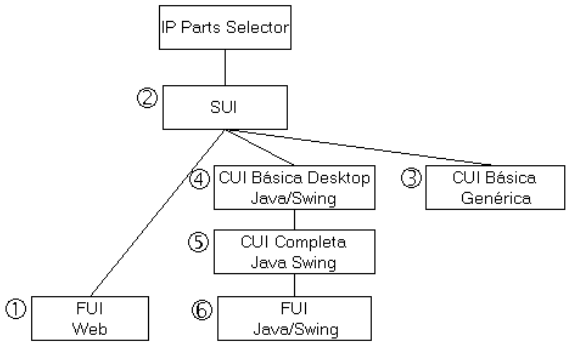


Figura 5. Exemplo de estrutura do CIP "Parts Selector"

5. Projetando interfaces a partir dos casos de uso

Casos de uso, hoje, são consensualmente aceitos como muito importantes para a modelagem de software, sobretudo orientado a objetos, e que está no núcleo de abordagens como UP (e RUP) e em notações como UML. Por definição, um caso de uso contém uma narrativa que descreve a interação que se dá pela troca (envio e/ou recebimento) de informações entre um ator e um sistema [6].

Em um processo de desenvolvimento dirigido por casos de uso, é a partir das descrições dos casos de uso que se desenvolve o projeto da IU. Atualmente, é senso comum de que estas descrições devem estar no grau de abstração essencial [7, 6]. Na narrativa essencial, são explicitadas as interações entre o ator e o sistema sem descrever detalhes de como e onde estas interações serão realizadas.

As seqüências de interações (ou ações) que compõem a narrativa do caso de uso não são identificadas como um elemento no metamodelo de casos de uso da UML. Entretanto, para aplicar a nossa abordagem de reuso, é necessário explicitar estas interações através da identificação dos fluxos e passos que compõem a narrativa do caso de uso. Desta forma, propomos que o metamodelo de casos de uso seja estendido conforme apresentado na Figura 6.

Figura 6. Metamodelo de Caso de Uso

A partir da modelagem dos casos de uso essenciais, constrói-se um protótipo da interface com o usuário. O protótipo é uma representação limitada (um croqui ou esboço) do projeto da interface que permite aos usuários ter uma visão concreta e visual da interface do sistema e explorar a sua conveniência [17].

Os protótipos constituem um conjunto de artefatos que chamamos de casos de uso apresentados. Os casos de uso apresentados são a forma visual e perceptível da descrição dos fluxos dos casos de uso essenciais. Eles ainda não fazem referência aos objetos do sistema, pois representam um nível intermediário entre a narrativa conceitual definida no modelo de casos de uso e os objetos e suas colaborações definidos na fase de realização dos casos de uso.

Somente após a conclusão da fase de apresentação dos casos de uso é que se pode começar com a atividade de projeto OO propriamente dito, onde os elementos da IU, juntamente com os conceitos, são traduzidos em termos de objetos e classes e define-se como esses objetos e classes colaboram entre si para a realização do caso de uso.

6. Padrões de casos de uso

Considerando que os casos de uso servem como base para o projeto de IU, parece uma abordagem natural considerar o reuso de casos de uso como uma estratégia para reuso de interfaces. Para isso, é preciso definir o que torna um caso de uso reusável. Dado que a probabilidade de algo ser reusado está diretamente relacionada com a forma de sua descrição, alguns autores [2] sugerem que casos de uso essenciais são mais reusáveis do que os casos de uso concretos ou do que cenários.

Na descrição dos casos de uso essenciais, é possível identificar padrões de casos de uso (use case patterns) [1, 19, 15] que podem ser documentados e catalogados de forma semelhante a padrões de projeto (design patterns) [12].

Logo, um padrão de caso de uso é aquele que pode ser reusado em função de representar uma tarefa (orientado a tarefa) ou um domínio (orientado a domínio) recorrente a vários sistemas.

Estes padrões de casos de uso representam soluções para problemas específicos e podem ser reusados em contextos similares. Procura-se, sempre que possível, definir padrões de casos de uso parametrizados, ou seja, a parametrização do texto da narrativa

do caso de uso de forma a aumentar as chances de reuso. Parâmetros tornam localizadas as referências específicas ao sistema ou ao domínio da aplicação para o qual ele foi criado e possibilita que o mesmo caso de uso possa ser usado em situações diferentes.

De uma forma geral, os padrões de casos de uso orientados a tarefas são mais facilmente parametrizáveis e reusáveis do que padrões de casos de uso orientados a domínios.

Os padrões de casos de uso são muito mais úteis se puderem ser relacionados uns aos outros e/ou agrupados por tarefas ou domínios em comum. Constantine e Lockwood propõem que, além dos relacionamentos de inclusão, extensão e generalização, os casos de uso sejam relacionados por afinidade (semelhança e equivalência) [7]. Além destes relacionamentos, os casos de uso que participam na solução de um determinado problema podem compor uma linguagem de padrões de casos de uso a qual chamamos de *padrão de domínio*.

Um padrão de domínio descreve o conjunto de casos de uso que são re-correntes na modelagem de casos de uso de sistemas de um mesmo domínio de problema e que se relacionam entre si e colaboram para dar suporte à prestação de uma funcionalidade de valor para os usuários. Um padrão de domínio de comércio eletrônico, por exemplo, pode conter os casos de uso: "Consulta um produto", "Busca por palavra-chave", "Busca por categoria", "Coloca no carrinho de compras", "Efetua compra", entre outros, pois estes casos de uso são comuns a todos os sistemas de comércio eletrônico. Da mesma forma, o conjunto de casos de uso que suportam o serviço de autenticação e de permissões de usuários, e o conjunto de casos de uso que suportam o serviço de publicação de notícias em um site na Web, são exemplos de padrões de domínio.

7. Ligando padrões de casos de uso a linguagens de descrição de interfaces com o usuário

Da mesma forma que um padrão concreto de interação, um padrão de caso de uso pode conter vários níveis de reificação de IU, formando uma hierarquia de SUIs, CUIs e FUIs que modelam e implementam a IU do padrão de caso de uso. Ao conjunto de artefatos que compõem a hierarquia juntamente com o padrão de caso de uso, chamamos de *padrão de caso de uso reificado (reified use case pattern - RUCP).*

De forma semelhante à estrutura de um CIP, a estrutura de um RUCP é apresentada na Figura 7. No RUCP, uma SUI é o protótipo que constitui a apresentação do padrão de caso de uso. Todos os demais elementos constituem os vários níveis de reificação deste protótipo.

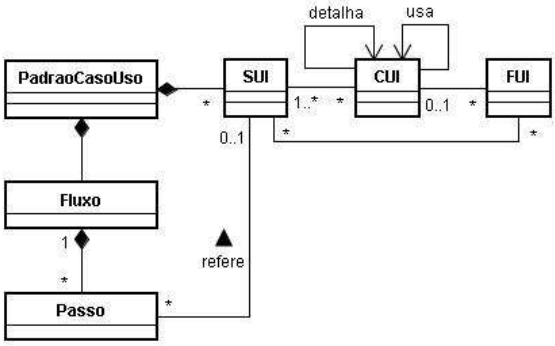


Figura 7. Estrutura de um padrão de caso de uso reificado.

Como o padrão de caso de uso deve ser escrito na forma essencial (independente de tecnologia), podem-se associar a este padrão hierarquias de IUs nos mais diversos tipos de ambiente de interação e de tecnologias de implementação. Um único padrão de caso de uso pode, por exemplo, conter SUIs e CUIs que o apresentem em um ambiente de interação Web, em um ambiente gráfico desktop, em um ambiente de um telefone celular, e assim por diante. Para cada um destes ambientes, podem existir FUIs que os implementem nas mais variadas linguagens de programação e/ou arquiteturas.

Cada passo do padrão de caso de uso pode ser associado com a respectiva IU que suporta a execução do passo. Isso permite "fracionar" a IU do caso de uso, restringindo-a ao conjunto de espaços de interação e de elementos de interação necessários à realização do passo.

É neste nível de granularidade da IU que se apresentam a maioria das soluções propostas pelos padrões de interação, permitindo - como veremos a seguir - o uso integrado e sistemático destes padrões no projeto e reuso de IHC.

8. Casos de reuso

Um caso de reuso existe quando são encontrados e aplicados artefatos para serem reusados a partir da especificação de um caso de uso. Na nossa abordagem, o principal elemento de ligação entre o caso de uso a ser desenvolvido e os possíveis artefatos a serem reusados é o padrão de caso de uso.

A abordagem utiliza a hierarquia de reificação do padrão de caso de uso como fio condutor do reuso. A identificação dos possíveis artefatos a serem reusados segue uma orientação top-down, ou seja, são analisa-

das as possibilidades de reuso a partir de especificações mais conceituais dos casos de uso (níveis mais altos da hierarquia de reificação) até as mais concretas (níveis mais baixos da hierarquia de reificação). A quantidade de artefatos reusados dependerá do acervo de artefatos disponível e da decisão tomada sobre qual o caminho a seguir no nível mais baixo.

As atividades para a identificação de artefatos reusáveis ocorrem de forma integrada com o processo de desenvolvimento OO. Estas atividades são chamadas de *análise de reuso de casos de uso* e *análise de reuso de interações*. A análise de reuso de casos de uso acontece na fase de análise de requisitos do SsD em paralelo às atividades de modelagem de casos de uso e de modelagem conceitual. Posteriormente, na fase de apresentação do caso de uso, ocorre a análise de reuso de interações em paralelo às atividades de projeto e de prototipação da IU. Estas atividades serão detalhadas a seguir.

8.1 Análise de reuso de casos de uso

Na análise de reuso de casos de uso procura-se identificar os padrões de domínio e padrões de casos de uso que sejam aderentes aos requisitos funcionais do SsD (Sistema sendo Desenvolvido). A pesquisa é feita com o enfoque tanto na busca de casos de uso com recorrência de domínio do problema quanto nos casos de uso com recorrência de tarefas padrão. O resultado desta atividade é um conjunto de casos de uso candidatos a reuso no SsD.

A seguir, para cada caso de uso candidato, procura-se identificar, na hierarquia de reificação desse caso de uso, até que nível de artefatos será possível reusar. Obviamente, poderão ser encontrados mais artefatos para reuso se, no momento da realização desta atividade, o ambiente de interação e a tecnologia de implementação do SsD já tiverem sido definidos. Caso contrário, o reuso poderá se restringir em nível de narrativa do caso de uso ou, no máximo, em nível de SUI.

8.2 Análise de reuso de interações

Durante a fase de apresentação dos casos de uso ocorre a segunda análise de reuso. Nesta fase, o foco é explorar as possibilidades de reuso de interações para todos os casos de uso do SsD em que não foi possível identificar, no mínimo, reuso em nível de SUI na análise de reuso anterior.

A análise de reuso de interações ocorre simultaneamente com as atividades de projeto e de prototipação da IU. À medida que o projetista de interface faz a leitura da narrativa do caso de uso e tenta esboçar uma IU que o suporte, também deve fazer uma pesquisa no repositório em busca de padrões concretos de interação (CIPs) que sejam adequados à execução das tarefas descritas nos passos do caso de uso. A busca no repositório deve ser feita de forma recursiva e incremental ao longo do projeto e prototipação da IU e deve levar em conta a natureza da tarefa, os requisitos de usabilidade e as informações necessárias para um passo ser executado.

Para cada CIP identificado como adequado para a IU que está sendo projetada verifica-se, na hierarquia de reificação deste CIP, até que nível e quais os artefatos que poderão ser reusados. O critério de escolha destes artefatos deve se basear na existência de um ramo na hierarquia do CIP que seja aderente ao ambiente de interação, tecnologia de implementação e compatibilidade com a arquitetura do SsD.

9. Conclusão

Neste artigo, foi apresentada uma abordagem de reuso que pode contribuir para o aumento do reuso de IUs durante o processo de desenvolvimento de um software. Nossa abordagem prevê o reuso de IUs a partir da especificação das interfaces de padrões de caso de uso, padrões de interação e dos artefatos de software que o implementam.

Desta forma, pretende-se atingir os seguintes objetivos: 1) propiciar o reuso de interfaces de usuário em vários níveis e 2) explicitar o comportamento das IUs como forma de garantir implementações consistentes e com alto grau de usabilidade. A definição e a construção das interfaces com o usuário ainda carece de abordagens efetivas e sistemáticas de reuso se comparadas com as práticas já existentes de reuso dos demais artefatos de software. Além disso, deve-se fazer um esforço para que a prática de reuso esteja adaptada e integrada com as demais atividades de projeto do software de forma que seja praticado naturalmente como parte integrante do processo de desenvolvimento. Como perspectivas de continuidade deste trabalho vislumbram-se: a) definição da estrutura e funcionamento de um repositório de CIPs e RUCPs, e b) a integração deste repositório com ferramentas CASE de modelagem da UML e ambientes integrados de desenvolvimento (IDEs).

10. Referências

[1] Biddle, R.; Noble, J.; Tempero, E. Essential Use Cases and Responsibility in Object-Oriented Development. 2001. Disponível em: <

http://www.foruse.com/articles/euc-responsibility.pdf >. Acesso em: abril 2006.
[2] Biddle, R.; Noble, J.; Tempero, E. Supporting Reusable Use Cases In Proceedings of the 7th International Conference on Software Reuse: Methods, Techniques, and Tools. pp. 210-226. 2002.

[3] Borchers, J. A Pattern Approach to Interaction Design em Proceedings of the conference on Designing interactive systems: processes, practices, methods, and techniques. Pp. 369-378. 2000.
[4] Borchers, J. Designing Interactive Music Systems: A Pattern Approach, in 8th International Conference on HCI. 1999. pp.276–280.
[5] Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouil-Lon, L., Vanderdonckt, J. A Unifying Reference Framework for Multi-Target User Interfaces. Interacting with Computers, vol. 15, no. 3, 289–308. 2003.
[6] Cockburn, A. Writing Effective Use Cases [S.l.]: Addison-Wesley, 2001.
[7] Constantine, L.; Lockwood, L. Software for Use: A Practical Guide to The Models and Methods of Usage-centered Design. [S.l.]: Addison-Wesley, 1999.
[8] Cox, B. "What if there is a silver bullet… and the competition gets it first?" em Dr. Dobb's Journal, Oct 1992.
[9] Duyne, D.; Landay J.; Hong, J. The Design of Sites: Patterns, Principles, and Processes for Crafting a Customer-Centered Web Experience. Boston: Addison-Wesley, 2002.
[10] Gaffar, A. et al. Modeling Patterns for Task Models, em TAMODIA 2004, pp 99-104. 2004.
[11] Gaffar, A.; Seffah, A.; Van de Poll, J. HCI Pattern Semantics in XML: a Pragmatic Approach. 2005.
[12] Gamma, E. et al. Padrões de Projeto – Soluções reutilizáveis de Software Orientado a Objetos. Bookman, 2000.
[13] Mahemoff, M.; Johnston, L. Pattern Languages for Usability: An Investigation of Alternative Approaches. In Tanaka, J. (Ed.), in APCHI 98 Proceedings , 25-31. 1998.
[14] Mahemoff, M.; Johnston, L. Usability Pattern Languages: the "Language" Aspect. em HCI: Interact '01. pgs 350-358. Disponível em http://mahemoff.com/paper/ Acesso em 04/2006.
[15] Overgaard, G.; Palmkvist, K. Use Cases Patterns and Blueprints. Indianapolis: Addison-Wesley, 2005.
[16] Pattern Language Markup Language (PLML) Disponível em www.hcipatterns.org. nov/2006.
[17] Preece, J.; Rogers, Y.; Sharp, H. Design de interação: além da interação homem-computador. Porto Alegre: Bookman, 2005
[18] Rijken, D. The Timeless Way... the design of meaning. SIGCHI Bulletin.Vol. 6, No. 3. PP. 70–79. 1994
[19] Saeki, M. Reusing Use Case Descriptions for Requirements Specification: Towards Use Case Patterns in Proceedings of the Sixth Asia Pacific Software Engineering Conference. 1999.
[20] Sindre G.1; Conradi R.; Karlsson E.-A The REBOOT Approach to Software Reuse Journal of Systems an Software, Volume 30, Number 3, September 1995, pp. 201-212(12)
[21] Sutcliffe, A. & Carroll, J. Designing Claims for Reuse in Interactive Systems Design, in International Journal of Human-Computer Studies, /50(3), pp 213-242. 1999.
[22] The Parts Selector Interaction Pattern. Disponível em www.welie.com/patterns/showPattern.php?patternID=parts-selector . Acessado 07/2006.
[23] Tidwell, J. Interaction Patterns In: Proceedings of Pattern Languages of Program Design. 1998.
[24] Todd, E.; Kemp, E.; Phillips, C. What makes a good User Interface pattern language? In: Proceedings of the 5th AUIC2004. 2004.
[25] Tracz, W. Software Reuse Myths, em ACM SIGSOFT Software Engineering Notes vol. 13 no. 1, Janeiro, págs. 17-21.
[26] UIML User Interface Markup Language Specification Working Draft 3.1. March 2004. Disponível em http://www.oasis-open.org/committees/download.php/5937/uiml-core-3.1-draft-01-20040311.pdf. Acesso 07/2006
[27] Vanderdonckt, J.; Bodart, F. Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection. In Proc. of ACM Conf. on Human Aspects in Computing Systems InterCHI'93 (Amsterdam, April 24-28, 1993). ACM Press, New York, 424–429. 1993
[28] Welie, M.; Trætteberg, H. Interaction Patterns In User Interfaces in Proceedings of the Pattern Languages of Programming PloP' 2000.Disponível em: http://www.idi.ntnu.no/~hal/publications/design-patterns/PLoP2k-Welie.pdf . Acesso 04/2006.
[29] Welie, M.; Veer, G.; Eliëns, A. Patterns as Tools for User Interface Design. In International Workshop on Tools for Working with Guidelines. pp. 313-324. 2000.

Modelado de Aplicaciones con Procesos Concurrentes y Distribuidos

Universidad Nacional de la Matanza - Depto. de Ingeniería e Investigaciones Tecnológicas
Florencio Varela 1903 (1754) San Justo - Buenos Aires - Argentina

Mg. Daniel A. Giulianelli
dgiulian@unlam.edu.ar

Ing. Rocío A. Rodríguez
rrodri@unlam.edu.ar

Ing. Pablo M. Vera
pablovera@unlam.edu.ar

Resumen

En este paper se presenta al DEA "Diagrama de Estados Activos", el cual fue desarrollado por el equipo de trabajo, para poder modelar todos los aspectos de las aplicaciones con procesos concurrentes y distribuidos. Como punto de partida se evaluaron los distintos diagramas de UML 2.0 y viendo que ninguno de ellos se adaptaba por completo a las necesidades presentadas, se elabora éste modelo que toma las características de algunos diagramas de UML (Lenguaje Unificado de Modelado) y agrega elementos necesarios para este tipo de aplicaciones. El DEA permite visualizar en un único diagrama aspectos que de modelarse con UML implicarían la construcción de varios diagramas y el uso de estereotipos.

1. Antecedentes

Con la intención de modelar aplicaciones con procesos concurrentes y distribuidos usando UML 2.0 (ver introducción [6] y [2]), nace en el año 2005 este trabajo de investigación. Viendo que para modelar dichas aplicaciones era necesario utilizar un conjunto de diagramas y estereotipos, surge la idea de desarrollar un modelo que permita unificar todos los conceptos necesarios para una correcta representación en un único diagrama. Producto de la interacción con pares de la disciplina y luego de varios trabajos preliminares surge dicho diagrama al que hemos denominado DEA (Diagrama de Estados Activos).

En octubre del 2005 el trabajo fue presentado, aceptado y expuesto, en el Congreso Argentino de Ciencias de la Computación desarrollado en la Universidad Nacional de Entre Ríos (CACIC 2005). Se continúo trabajando y los resultados de dichos avances fueron presentados, aceptados y expuestos en la Jornada de Jóvenes Investigadores de Universidades Nacionales organizada por la Universidad Nacional de San Luís (JI2005). Para comprender que aspectos de UML se toman en cuenta para la propuesta que presentamos, sugerimos leer el paper correspondiente a la primera versión de este trabajo presentado en el CACIC 2005, en donde se presenta una introducción sobre los aspectos más relevantes de UML que fueron considerados para la construcción del DEA [8].

Después de estas presentaciones habiendo analizado las posibilidades que presenta UML 2.0 para modelar este tipo de aplicaciones y habiendo realizado una importante cantidad de modelados por medio del DEA encontramos más que necesario continuar enriqueciendo nuestra propuesta, dotando al DEA de recursos los que a nuestro criterio simplifican, facilitan y enriquecen el desarrollo de modelos.

A fin de poder analizar la evolución de nuestro trabajo actual, se encuentran publicadas las mejoras realizadas en las distintas versiones del mismo [10].

Uno de los objetivos del presente trabajo es presentar las mejoras realizadas sobre el DEA y realizar una comparativa de las ventajas de nuestra propuesta con respecto a UML 2.0. Con este fin se presenta el modelado de una misma aplicación utilizando ambas metodologías.

2. Introducción

Tomando como referencia a UML nos propusimos construir un Diagrama que reuniera las características necesarias para poder con él modelar procesos concurrentes y distribuidos.

Para ello tomamos las particularidades del Diagrama de Transición de Estados (que cubren la vista dinámica de un sistema), las correspondientes al Diagrama de Actividades (que muestra el flujo de control entre actividades), algunas características del Diagrama de Despliegue (que cubre la vista estática de un sistema) e incorporando características propias de los sistemas con procesos concurrentes y distribuidos, construimos un modelo al que hemos denominado "Diagrama de Estados Activos (DEA)" (ver Figura 1).

Figura 1: Elementos del DEA

Este modelo que proponemos lo consideramos adecuado para:

1) Representar un proceso distribuido: Se utilizan las calles que caracterizan al clásico Diagrama de actividades [7] [14], para determinar en que nodo se realizará la ejecución de ciertos procesos. Para aquellos casos donde el medio de comunicación entre los nodos sea de relevancia, se indicará con una línea entre ambos el tipo de vínculo utilizado, tal como se haría en UML en un Diagrama de Despliegue.

2) Representar la concurrencia de procesos: Se utilizan las líneas de sincronismo del Diagrama de actividades.

3) Detallar la información de los estados: Además de indicarse el nombre del estado se puede detallar como en el clásico DTE acciones de entrada y salida, transiciones internas y eventos diferidos.

4) Mostrar el cambio de estados: Se utilizan las transiciones del DTE indicando la o las condiciones, así como el evento y en el caso que existan, las acciones que permiten el paso de estado.

2.1. Particularidades de la propuesta

Del análisis del modelo presentado surgen las siguientes conclusiones:

1) Al agregar al clásico Diagrama de Transición de Estados las calles, queda indicado si el proceso comenzado en cierto estado al pasar a otro involucra o no un cambio de nodo. Es decir que al cumplirse cierta condición podrá continuarse el proceso en otro nodo (calle). Es importante destacar que la acción de cambio de nodo puede no estar asociada a una condición, en ese caso el cambio de nodo se producirá sin evaluar condiciones.
2) Las líneas de sincronismo pueden compartirse entre varios nodos por lo tanto el flujo del procesamiento se continúa en un nodo específico cuando este obtenga una respuesta de los otros nodos que están procesando en forma paralela, ya sea por necesitar un resultado o por la finalización de los mismos

2.1.1. Particularidades del modelado de procesos distribuidos. El procesamiento distribuido requiere que los procesos alojados en distintos host se comuniquen de alguna manera para poder intercambiar información. Existen dos formas distintas de realizar dicha comunicación: mediante el envío de mensajes y por medio de la utilización de RPC (Remote Procedure Call – Llamadas a procedimientos remotos).

A su vez los mensajes pueden ser asincrónicos o sincrónicos. Un mensaje sincrónico obliga al emisor a esperar una respuesta antes de continuar con sus tareas. Por el contrario si es asincrónico se envía el mensaje y se continúa con el resto de las tareas. Cabe destacar que los RPC son en su mayoría llamadas sincrónicas ya que actúan como simples llamadas a procedimientos como si estuvieran en una misma computadora.

Para modelar la comunicación entre los procesos se mantiene la nomenclatura habitual de UML donde un mensaje sincrónico se representa con una punta de flecha rellena y un mensaje asincrónico con una punta flecha abierta (es recomendable consultar el manual de especificación de UML 2.0 [12]). Respetando esta nomenclatura proponemos para mayor claridad diferenciar los RPC de los mensajes usando la nomenclatura mostrada en la Figura 2.

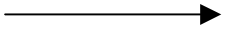
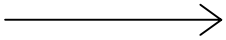
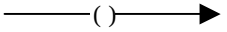
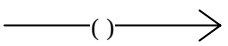
Mensaje Sincrónico	
Mensaje Asincrónico	
RPC Sincrónico	
RPC Asincrónico	

Figura 2: Nomenclatura mensajes y RPC

Los nodos que interactúan pueden tener diferentes tipos de conexiones físicas entre sí, lo que en algunas

ocasiones es importante destacar, ya que según sea el tipo de enlace habrá ciertas velocidades de transferencia que variarán. Lo que puede permitir decidir derivar ciertos procesos a un determinado nodo u a otro, o manejar distintos tiempos de time out para las respuestas. Al igual que en el diagrama de despliegue de UML se indicará en el DEA través de una línea entre cada par de nodos la característica del enlace.

2.1.2. Particularidades del modelado de procesos concurrentes. El modelado del procesamiento concurrente puede requerir la diferenciación de los distintos hilos (threads) de ejecución del sistema. Si bien la propuesta al incorporar las líneas de sincronismo del diagrama de actividades ya muestra los procesos o hilos que se ejecutan en paralelo, se propone también para una notación más clara en casos en los que se detallen varios estados dispares dentro de cada hilo. Se propone utilizar una notación de calles con líneas punteadas dentro de la calle principal del nodo para indicar el procesamiento independiente de cada hilo (ver Figura 3).

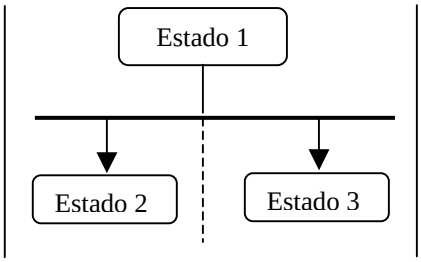


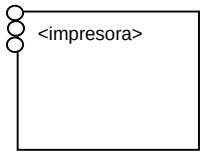
Figura 3: Subcalles para los hilos de ejecución

Cuando se utilizan hilos es muy posible que existan recursos que se deben compartir y por lo tanto es necesario administrar su acceso, ya que solamente un thread puede utilizarlo en un momento dado. Dos métodos comunes para la administración de recursos en un ambiente concurrente son los semáforos y los monitores [4].

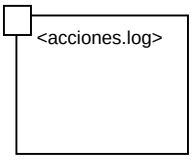
1) Los semáforos son variables especiales que señalizan el acceso a los recursos utilizando dos primitivas (ejemplo: signal y wait) que retardan momentáneamente el acceso a un recurso en uso.

2) Los monitores son facilidades que brindan los lenguajes de programación y cumplen con la misma función que los semáforos pero de una forma más transparente ya que no es necesario manejar en forma manual un envío de primitivas signal y wait desde los procesos sino que esto se maneja en forma automática al encolar un proceso en el recurso controlado por el monitor.

Proponemos dos construcciones gráficas para indicar recursos compartidos y el método de acceso a los mismos. Dentro de estas construcciones es posible especificar el tipo de recurso al que nos estamos refiriendo (ver Figura 4).



Recurso compartido administrado mediante un semáforo.



Recurso compartido administrado mediante un monitor.

Figura 4: Nomenclatura designada para Semáforos y Monitores

2.1.3. Cardinalidad. Cuando se cuenta con múltiples conexiones provenientes desde distintos nodos, generalmente existe un proceso principal que monitorea las conexiones y al llegar nuevas conexiones crea distintos hilos para cada una de ellas. A menudo, las acciones que realizan cada uno de esos hilos son idénticas, por lo tanto proponemos la utilización de una nomenclatura, similar a un objeto con varias ocurrencias en UML, pero aplicado a las calles que está representando el hilo en ejecución (ver Figura 5).

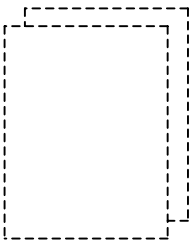


Figura 5: Las acciones que se realizan "n" veces se indican dentro de esta representación gráfica.

A continuación se presenta la Figura 6 en la que puede observarse el modelado de una aplicación por medio del DEA. Esta aplicación consiste en un juego online multijugador. Por cada Jugador que se conecta al servidor, deben ocurrir una serie de estados ("Recibiendo conexión", "Leyendo configuración" y "Registrando jugador") que se realizarán de la misma forma para cada uno de ellos.

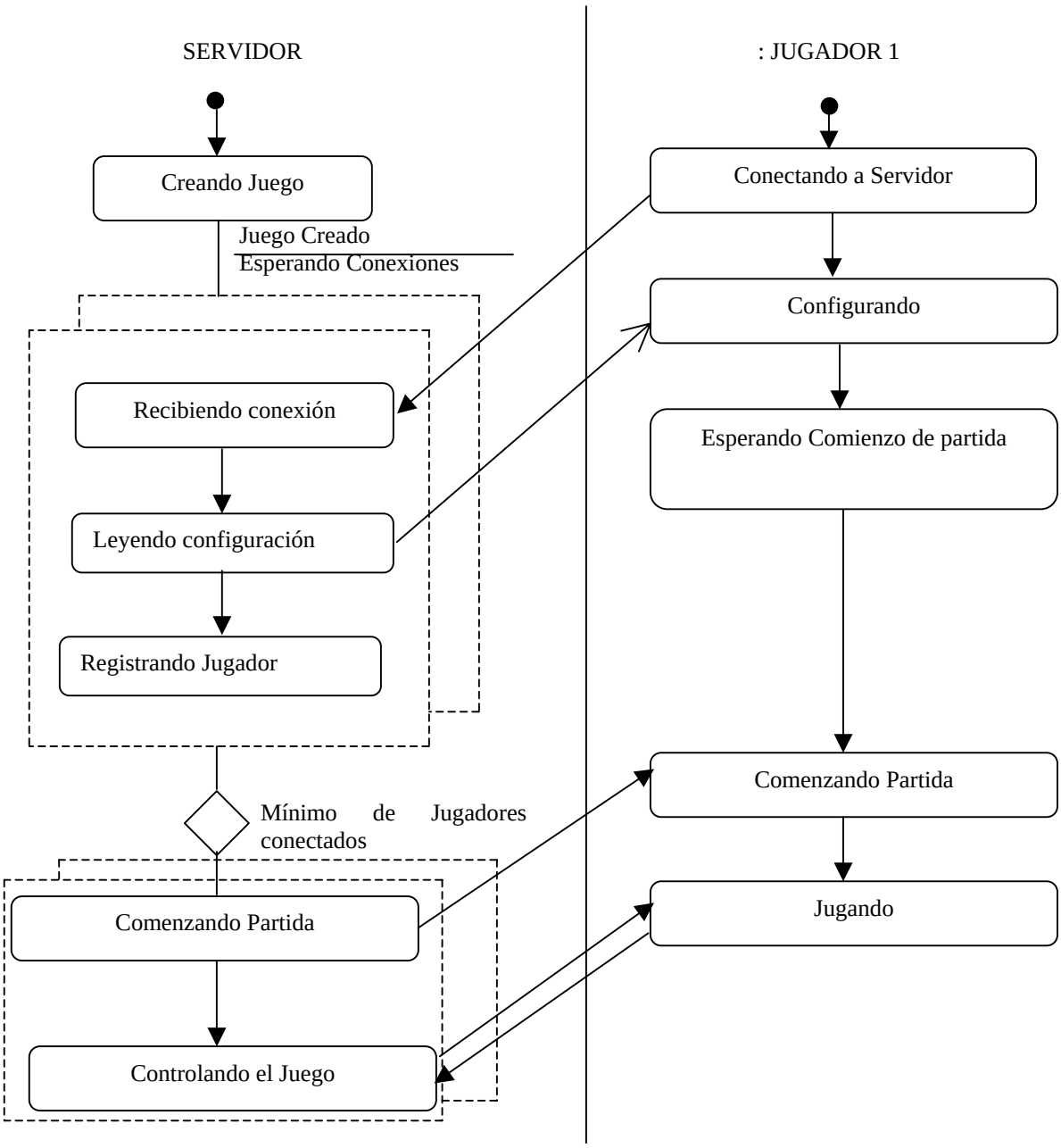


Figura 6: Modelado a través del DEA de un juego online multijugador.

3. Metodología de Modelado

Se propone a continuación un listado de pasos que permitirán realizar un Diagrama de Estados Activos, de forma sistemática.

1) Identificar los distintos nodos o locaciones donde se va a ejecutar la aplicación para así definir con cada una de ellos las calles del diagrama.

2) Identificar el nodo desde el cual se va a iniciar el proceso o la funcionalidad que estamos modelando y ubicar la calle que lo representa en el centro del diagrama para minimizar el cruce de líneas entre las demás calles ya que es usual que éste sea el nodo principal que comanda al resto.

3) Determinar cuales conexiones entre nodos son relevantes de destacar e indicar entre dichos nodos el tipo de conexión física. Solo se deberán marcar las conexiones entre los nodos que tengan cierta relevancia como por ejemplo que puedan tener problemas de velocidad, ancho de banda, etc.

4) Comenzar el modelado desde el nodo ubicado en la calle central, usando la nomenclatura habitual del Diagrama de Transición de estados empezando con el círculo lleno que indica el comienzo del proceso.

5) Continuar el modelado utilizando de forma habitual los estados, cambios de estado y transiciones del DTE pero teniendo en cuenta que ahora podemos utilizar la nomenclatura del diagrama de actividades cuando tenemos actividades que se realizan en paralelo.

6) Cuando el flujo de control principal o uno de los flujos secundarios salen del nodo actual utilizar la nomenclatura de mensajes o RPC propuesta para clarificar de que forma se realiza dicha comunicación.

7) Si uno de los estados requiere la utilización de un recurso compartido especificar como se realiza el control de acceso al mismo mediante un semáforo o un monitor.

8) Para rutinas repetidas dentro de un mismo nodo, es decir que disponen de un control de conexión y para cada una de ellas se realiza un proceso igual, independientemente de que nodo venga la conexión, englobar dicha funcionalidad dentro de un recuadro de cardinalidad.

4. Modelado de una aplicación

Se desea modelar el control de acceso a una bóveda de alta seguridad.

Esta cuenta con diferentes mecanismos para controlar el acceso a la misma.

Como primera medida de seguridad se requiere poseer la llave que habilita un teclado mediante el cual se ingresa una clave de acceso. Existe un número de veces que se puede ingresar en forma errónea la clave antes de ser disparada la alarma. Una vez dado por válido dicho código se realiza un escaneo de retina para autentificar a la persona.

Luego se habilita un teclado en cada una de las dos sucursales, en donde se ingresa una clave de seguridad. Una vez chequeada la validez del código se procede a realizar un escaneo de retina a quienes ingresan dichas claves. Estos escaneos son procesados mediante los patrones almacenados en el servidor de la casa central. Si son válidos se habilita el acceso a la bóveda.

4.1. Modelado por medio de UML

Se realizarán a continuación una serie de diagramas para modelar los aspectos más relevantes de la aplicación:

1) Diagrama de despliegue
2) Diagrama de clases
3) Diagrama de colaboraciones

Se recomienda leer una breve reseña de la construcción de los modelos que se emplearán (ver [11] y [13]).

4.1.1. Diagrama de Despliegue. Mediante el diagrama de despliegue se pueden observar los distintos nodos, el tipo de conexión que existe entre ellos y las operaciones que despliega cada uno de esos nodos (Ver Figura 7).

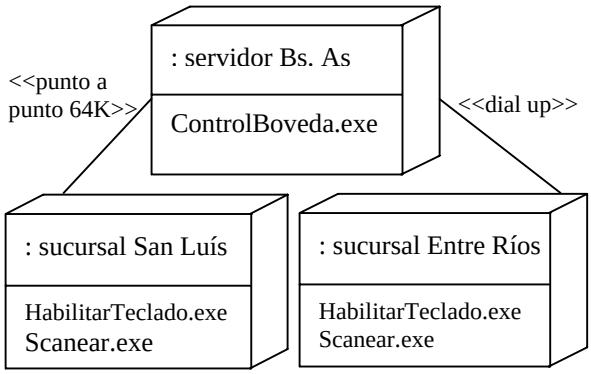


Figura 7: Diagrama de Despliegue

Como se observa en la figura anterior, este diagrama muestra la configuración en tiempo de ejecución de los nodos de procesamiento y los componentes que residen en ellos.

4.1.2. Diagrama de Clases. Este diagrama permite identificar las clases, las clases activas y las operaciones que contienen las mismas (ver Figura 8).

Una clase activa es una clase cuyos objetos tienen uno o más procesos o hilos de ejecución por lo tanto pueden dar lugar a actividades de control. Es importante destacar que en UML las clases activas se denotan con el contorno grueso.

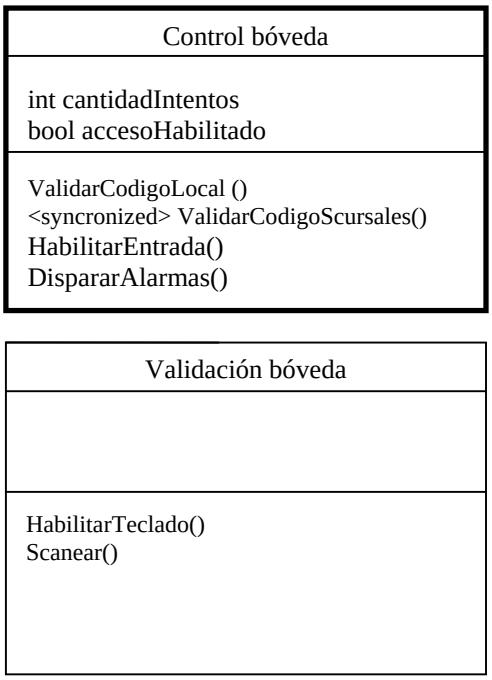


Figura 8: Diagrama de Clases

Como se observa en la figura anterior, este diagrama se utiliza para modelar la vista de diseño estática de un sistema la cual soporta los servicios que el sistema debe proporcionar a los usuarios finales.

4.1.3. Diagrama de Colaboraciones. En este diagrama que resalta la organización estructural de los objetos que envían y reciben señales se muestran las interacciones organizadas alrededor de las instancias y los enlaces de unas con otras.

En este caso fue imprescindible utilizar estereotipos para poder vincular éste diagrama con el de despliegue. Puede observarse que a través del estereotipo Location se logra identificar al nodo en cuestión (ver Figura 9).

Las instancias de las clases activas se diferencian de las instancias de clases no activas representándolas con el contorno grueso.

3) Se marcan las conexiones entre los nodos por medio de líneas y sobre cada una de ellas se indica el tipo de conexión. Entre Casa Central y la sucursal San Luís la conexión tal como lo mostramos en el diagrama de despliegue es punto a punto y de Casa Central a la sucursal Entre Ríos es Dial Up.

4) Tomando como punto de partida el nodo central, se indica el estado de inicio utilizando el círculo vacío (usando la nomenclatura habitual del Diagrama de Transición de estados).

5) Se comienza a modelar un diagrama de estados dentro del nodo central y recurrimos a una línea de sincronismo en el momento en que éste nodo se conecta simultáneamente con los nodos remotos.

6) Cuando se realiza una comunicación desde el nodo central hacia los otros y viceversa, usamos la nomenclatura propuesta para indicar los mensajes y

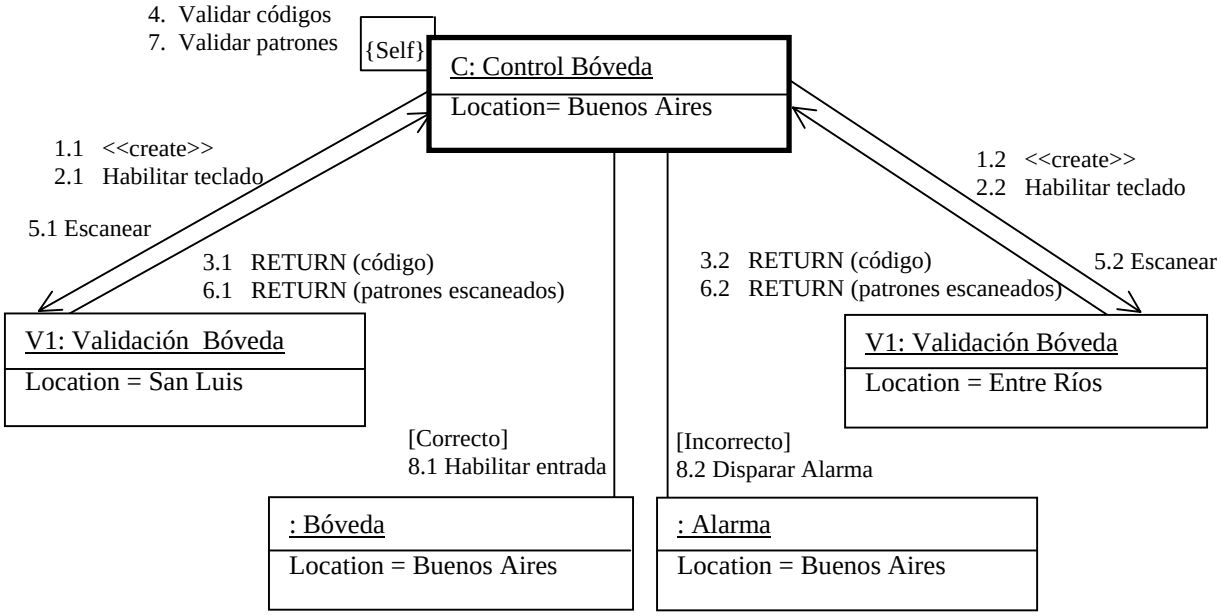


Figura 9: Diagrama de Colaboraciones

4.2. Modelado por medio del DEA

Se aplica la metodología de modelado indicada en la Sección 3, para modelar la aplicación descripta, por medio del DEA (Figura 10).

1) Se identifican los distintos nodos: Sucursal de San Luís, Sucursal de Entre Ríos y Casa Central en Buenos Aires. Entonces se consideran tres calles una para cada nodo.

2) Se ubica en el centro del diagrama la calle correspondiente al nodo Casa Central ya que desde allí se inicia el proceso o la funcionalidad que estamos modelando. Esto evitará el cruce de líneas lo que quitaría claridad al diagrama.

llamadas a procedimientos remotos (RPC) sincrónicos ó asincrónicos.

7) Si suponemos que las claves de las sucursales, San Luis y Entre Ríos se chequean contra un archivo que se encuentra en la Casa Central, ese archivo será un recurso compartido por lo cual utilizaremos por ejemplo un semáforo.

8) En esta aplicación cada sucursal ejecuta en Casa Central el control del ingreso de la clave y el chequeo de la cantidad de intentos fracasados. Esta rutina se englobará mediante la representación gráfica sugerida para denotar cardinalidad, a fin de mostrar que ambos nodos la realizarán en forma independiente de lo sucedido con el otro nodo.

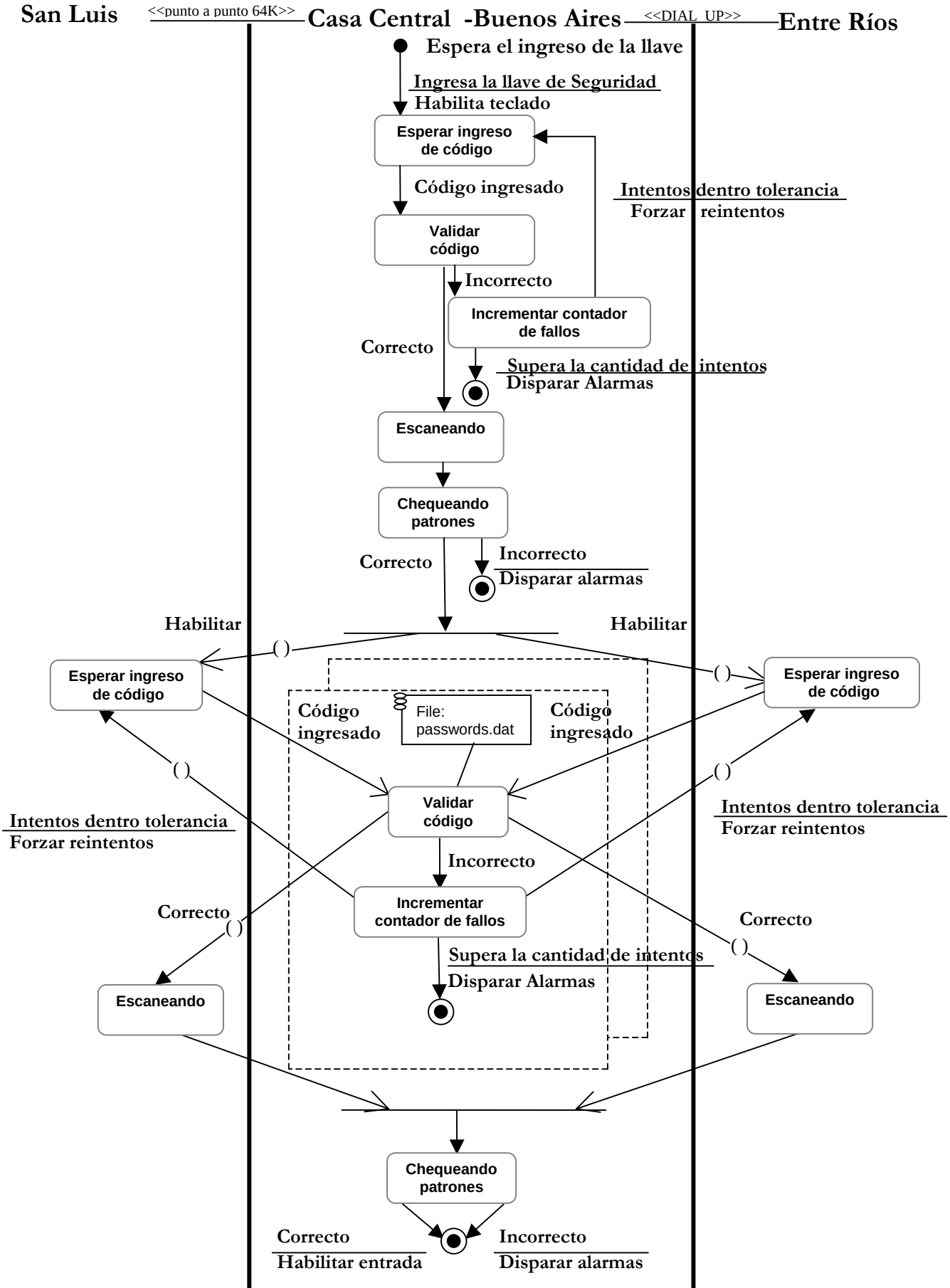


Figura 10: DEA que integra los elementos típicos anteriormente planteados para las aplicaciones con procesos concurrentes y distribuidos.

5. Comparación con UML

UML brinda herramientas para el modelado de procesos concurrentes mediante el uso de clases activas, dichas clases indican que sus objetos contienen diferentes hilos de ejecución.

Para poder modelar la comunicación entre diferentes procesos y threads UML propone la utilización de diagramas de objetos adornados con varios estereotipos como ser location que nos indica la ubicación física de ese hilo o proceso. También para establecer la ubicación física es posible realizar un diagrama de despliegue que indique que procesos ejecuta cada nodo y cuales son las conexiones físicas entre ellos. Si se quiere modelar como se comunican procesos ubicados en distintos nodos es posible que un diagrama de colaboración donde cada objeto tenga que estar estereotipado con la propiedad location se vuelva muy difícil de seguir. Por ello la propuesta del DEA es utilizar las calles para ver que es lo ocurre dentro de cada nodo y como se comunican unos con otros.

Para la comunicación entre procesos UML propone la diferenciación entre mensajes sincrónicos y asincrónicos. Esta metodología no especifica una semántica particular para llamadas a procedimientos remotos RPC. Estos últimos indicarían que en un momento dado se invoca un procedimiento ubicado en otro nodo sin que en dicho nodo necesariamente se este ejecutando un proceso que estuviera siendo monitoreado a la espera de mensajes. El párrafo precedente justifica porque resulta necesario diferenciar mensajes de llamadas a procedimientos remotos (RPC).

En cuanto a la utilización de diagramas de estado UML propone su utilización para modelar el comportamiento de cada una de las clases activas por separado. Siendo necesario para ver su interacción construir un diagrama de colaboración.

Para los recursos cuyo acceso debe ser sincronizado, UML propone estereotipar los métodos que por ejemplo deban ser sincronizados dentro de una clase pero no hace referencia a elementos físicos compartidos como ser un archivo, una unidad de disco, etc.. Elementos que muchas veces es necesario destacar para apreciar como varios hilos van a competir por dicho recurso.

6. Conclusiones

De la comparativa presentada en el Item 5, puede desprenderse que el DEA es una herramienta que permite modelar aplicaciones con procesos concurrentes y distribuidos, plasmando las características de estas aplicaciones en un único diagrama. Si bien el diagrama parece a simple vista ser más complejo (Figura 10), evita tener que realizar una serie de modelos (ver Figura 7, 8, 9) los cuales traen aparejados los siguientes inconvenientes:

1) Tener que relacionar los distintos modelos para lograr llegar a la visión que se observa en el DEA.

2) Tener que usar estereotipos para poder modelar características no nomencladas en UML.

3) El conjunto de modelos realizados en UML no logra mostrar la diferencia entre mensajes y llamadas a procedimientos remotos (RPC). Así como tampoco se pueden visualizar rutinas repetitivas, es decir el concepto de cardinalidad.

Por los motivos expuestos sostenemos que el DEA ayuda al análisis, incorporando las características de los modelos de UML y agregando aquellas características específicas de este tipo de aplicaciones, no soportadas por dichos modelos.

Lo expuesto nos motiva a proponer este modelo para presentarlo a un usuario final ya que consideramos que aporta una integración de las distintas visiones conseguidas a través de varios diagramas de UML.

Consideramos por último que el Diagrama de Estados Activos (DEA) puede transformarse en una herramienta a través de la cuál un usuario no experto en desarrollo de sistemas pueda con mínimas explicaciones, comprender y transmitir a sus pares el funcionamiento de una aplicación que requiere la utilización de procesos concurrentes y distribuidos.

7. Referencias

7.1 Libros

[1] Booch G, Rumbaugh J y Jacobson I. El lenguaje unificado de modelado. Addison Wesley, pp: 392-394, 2003.

[2] Fowler M. UML Distilled. Addison Wesley, Third Edition, Pearson Education , pp: 1-16, 2004.

[3] Kendall S. Fast Track UML 2.0, Apress, California 2004.

[4] Stalling W. Operating Systems Internals and Design Principles. Third Edition, Prentice Hall., pp: 208-230, 1998.

7.1 E-Book

[5] OMG Unified Modeling Language Specification. Versión 1.5 Marzo 2003.

7.2 Páginas Web

[6] http://es.tldp.org/Tutoriales/doc-modelado-sistemas-UML/multiple-html/

[7] http://www.creangel.com/uml/actividad.php

[8]http://ingenieria.unlam.edu.ar/contenidos/investigaciones/wisbd022.pdf

[9] http://www.investigamos.com.ar

[10] http://www.investigamos.com.ar/proyectos.htm#evolucion

[11] http://www.microsoft.com/spanish/msdn/articulos/archivo/230801/voices/modelsoftware.asp

[12] http://www.uml.org/#UML2.0

[13] http://www-gris.det.uvigo.es/~avilas/UML/node22.html

[14]http://www-gris.det.uvigo.es/~avilas/UML/node46.html

Requisitos No Funcionales: Evaluando el Impacto de las Decisiones

Marcela Quispe-Cruz
Escuela Profesional de Ingeniería de Sistemas
Universidad Nacional de San Agustín
Arequipa-Perú
marcela@spc.org.pe

Nelly Condori-Fernández
Departamento Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Valencia-España
nelly@dsic.upv.es

Resumen

Actualmente, no existe un consenso en la comunidad para la representación y tratamiento de los Requisitos No Funcionales. Sin embargo, estos requisitos son cruciales por el soporte que brindan a los requisitos funcionales repercutiendo en el éxito del proyecto. En el presente trabajo se presenta una especificación orientada a metas, basado en el lenguaje GRL, la noción de grados de satisfacción de dichas metas constituye la base de este trabajo. La evaluación del impacto de decisiones consiste en la evaluación de metas difusas e interdependencias que determinan el grafo de requisitos no funcionales. El objetivo es semi-automatizar este proceso de evaluación, mediante la definición de reglas expresadas en términos de reescritura.

1. Introducción

Los "Requisitos no Funcionales"[1] juegan un rol crítico durante el desarrollo de los sistemas de software ya que enfoca tanto aspectos de calidad como: la reusabilidad, mantenibilidad, seguridad, transportabilidad, exactitud, etc. y restricciones bajo el cual el sistema necesita operar. Por tal razón, dado que los Requisitos No Funcionales (RNFs) son igual de relevantes que los aspectos funcionales del sistema, en estos últimos años, se ha visto una mayor preocupación por tomar en cuenta algunas deficiencias que se presentan en los RNFs, tales como:

- *A* la *falta de un consenso para la representación de RNFs:* Se tiene trabajos basados en estereotipos UML y OCL para lograr una representación notacional [15], [5]. Así mismo, en [13] los RNFs son representados por medio de un grafo de *"softgoals"* e *"interdependencias"*. Entre las especificaciones formales orientadas al producto tenemos a NO-FUN [7] y QRL [14]. Siendo QRL, una especificación formal con una orientación semi-cualitativa.
- *En cuanto a un marco que soporte la integración de requisitos no funcionales con requisitos funcionales:* El trabajo de Cysneiros *et al* [4], proponen una integración por medio del uso de lexicones (LEL[11]) entre los RNFs y los modelos conceptuales.
- *Escaso número de herramientas de soporte en comparación a las existentes para requisitos funcionales:* SYBIL[10], es un sistema que provee un entorno para usar DRL (Lenguaje de Representación de Decisiones), maneja relaciones de metas y alternativas de decisiones, promueve la reusabilidad y evalúa la satisfacción de la meta. La herramienta OME[17] soporta los marcos metodológicos i* y NFR[3] además de ofrecer soporte para construir el grafo de metas difusas e interdependencias. NFR-Assistant[1], es un prototipo de herramienta CASE colaborativa que también soporta el marco metodológico NFR, permitiendo la evaluación del nivel de éxito de los RNFs y muestra el efecto de cada meta y contribución, la facilidad de aprendizaje para usar esta herramienta es analizada como ejemplo práctico en este artículo.

[1] Adoptamos el término de Requisitos No Funcionales por ser el más utilizado en la comunidad. Aunque los autores no están de acuerdo en su totalidad con dicha denominación.

Cada una de estas propuestas, pueden ser clasificados como orientados al producto u orientados al proceso. Además, cabe indicar que estamos de acuerdo con [13] y [7] en que ambas técnicas no deben ser consideradas como alternativas si no que deben ser complementarias.

El presente trabajo es orientado al proceso y está basado principalmente en el marco metodológico propuesto por Lawrence Chung *et al*[3]. Este marco permite tratar a los RNFs como metas difusas[2] y las interacciones que existen entre este tipo de metas (interdependencias). Básicamente la noción de **satisfacción** de meta es el término sobre el cual se construye dicho marco. La evaluación de metas difusas e interdependencias determinan el impacto de decisiones en el logro de los RNFs. Para la representación notacional de los RNFs, utilizamos un lenguaje de requisitos orientado a metas denominado GRL propuesto por [8]. El objetivo de este trabajo, es semi-automatizar el proceso de evaluación de impacto de decisiones, mediante la definición de unas reglas expresadas en términos de reescritura.

El artículo es estructurado de la siguiente manera: En la sección 2, se presenta un esquema general, donde se describe los componentes principales de la arquitectura propuesta, haciendo hincapié en conceptos genéricos de GRL para la representación de los RNFs. En la siguiente sección, se presenta brevemente conceptos generales acerca de los sistemas de reescritura. En la sección 4, explicamos detalladamente el proceso de evaluación, mediante la definición de un sistema de reescritura para la evaluación del grafo de metas difusas e interdependencias. En la sección 5, se presenta un caso de estudio elaborado con la finalidad de verificar las reglas propuestas. Finalmente, conclusiones y trabajos futuros.

2. Un Esquema General de RNFs orientado al proceso

En la figura 1, se muestra un esquema general propuesto para la administración de RNFs. Principalmente comprende tres bloques principales:

- Representación de los RNFs a través de un grafo de metas difusas.
- Definición de un conjunto de catálogos que permitan estructurar y almacenar tanto RNFs como métodos y reglas de correlación.
- Evaluación de los RNFs.

Este trabajo se centra principalmente en este último bloque, el cuál será explicado más detalladamente en una sección posterior.

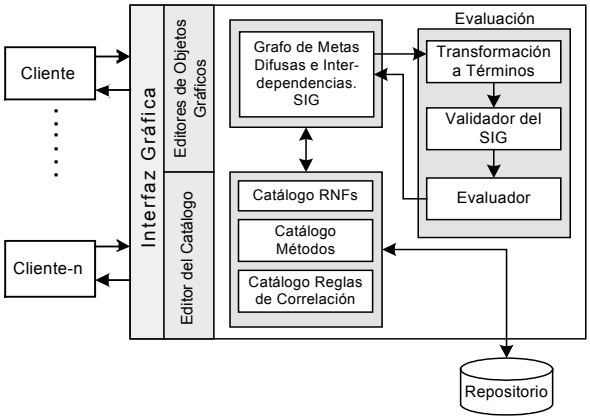


Figura 1. Esquema general de RNFs

2.1. Representación de los RNFs

Para la representación de los RNFs, utilizamos un lenguaje de requisitos orientado a metas conocido como GRL [8]. Este lenguaje provee de varios constructores para representar varios tipos de conceptos que aparecen durante el proceso de requisitos. Comprende de tres principales categorías: elementos intencionales (metas, tareas, metas difusas, recursos), relaciones intencionales (descomposición, contribución, correlación, dependencia) y actores.

Para la construcción del grafo de metas difusas e interdependencias [3] los elementos y las relacionales intencionales necesarias son:

2.1.1. Meta Difusa: Es una condición o estado de acontecimientos o eventos en el mundo que el actor podría lograr, pero a diferencia del concepto meta, la condición es lograda con diferentes niveles de satisfacción. La notación gráfica es:

```
<Meta difusa> ::= <Símbolo>
                  CONTIENE <Nombre> DE
                  <Tópico> [Atributos]
<Símbolo>::=
```

[2] metas difusas es la traducción al español que hemos dado al término original: *softgoals*.

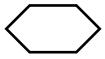
```
<Atributos>::= Denegado |
               Débilmente Denegado |
               Débilmente Satisfecho |
               Satisfecho |
               Conflicto | Indefinido
```

Un atributo para el caso de los RNFs será el valor que expresa el grado de satisfacción de la meta. Los valores definidos son:

- Denegado (D): Representa un grado de satisfacción negativo absoluto.
- Débilmente Denegado (W-): Representa un grado de satisfacción negativo pero relativo.
- Débilmente Satisfecho (W+): Representa un grado de satisfacción positivo pero relativo.
- Satisfecho (S): Representa un grado de satisfacción positivo absoluto.
- Conflicto (C): Representa un grado de satisfacción positivo y negativo al mismo tiempo.
- Indefinido (U): No presenta ni un grado de satisfacción positivo ni negativo.

2.1.2. Tarea u Operacionalización: Una tarea, llamada también operacionalización, es una manera particular de hacer algo. Las tareas pueden ser vistas como soluciones del sistema, las cuales satisfacerán a las metas difusas. La notación gráfica es:

```
<Operacionalización> ::= <Símbolo>
       CONTIENE <Nombre> <Atributo>
<Símbolo> ::=
```



```
<Atributo> ::= Denegado |
               Débilmente Denegado |
               Débilmente Satisfecho |
               Satisfecho | Indefinido
```

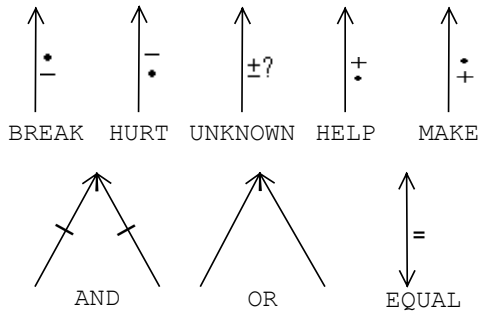
A diferencia de las metas difusas, para este nivel del grafo no se considera el atributo "conflicto" como valor de entrada al grafo. Ya que en este nivel, los diseñadores en el peor de los casos podrá entrar a un caso de indefinición, es decir, no tenga una decisión de aceptación o de rechazo por alguna alternativa de solución que satisfaga a algún conjunto de RNFs

2.1.3. Descomposición: Relación de conexión entre un componente y su sub-componente. La descomposición permite mostrar los componentes esenciales que necesitan ser logrados para la realización de una tarea. La notación gráfica es:

```
<Descomposición> ::= <Elemento
                          Descompuesto>
     { ES CONECTADO A  <Sub-Elemento>
       POR <Enlace de Descomposición>}
<Enlace de Descomposición> ::=
```

2.1.4. Contribución: Describe como un elemento intencional (meta difusa, operacionalización) contribuye para la satisfacción de otro elemento intencional. Los tipos de contribución que se presentan son: AND, OR, MAKE, BREAK, HELP, HURT, EQUAL, UNKNOWN. Su notación gráfica es:

```
<Contribución>::=<Elemento contribuyente>
     {ES CONECTADO A  <Elemento superior>
     POR <Enlace de contribución>}
<Enlace de contribución > ::=
```

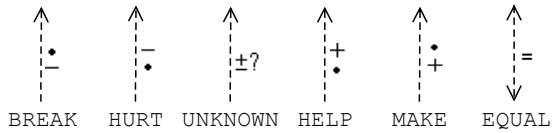


- Make: La contribución del elemento contribuyente es positiva y suficiente.
- Break: La contribución del elemento contribuyente es negativa y suficiente.
- Help: La contribución del elemento contribuyente es positiva pero no suficiente.
- Hurt: La contribución del elemento contribuyente es negativa pero no suficiente.
- Equal: Hay igual contribución en ambos sentidos
- Unknown: Hay alguna contribución desconocida.

2.1.5. Correlación: Las correlaciones permiten expresar conocimiento sobre interacciones entre los elementos intencionales. Un enlace de correlación es igual que un enlace de contribución sólo que no expresa un deseo explícito, pero es un tipo de efecto secundario.

```
<Correlación>::=<ElementoA> ES CONECTADO
                A  <ElementoB> POR
                <Enlace de correlación>
```

<Enlace de correlación> ::=



Cada interdependencia ("lazo") tiene una contribución o correlación, el cual indica el impacto local de un "nodo".

Una vez determinado los elementos y relaciones intencionales del grafo, resulta necesario proveer al diseñador o analista, mecanismos que faciliten la construcción del grafo. El proceso de catalogación, ayuda a disminuir las inconsistencias que se puedan presentar para el diseño del SIG (Soft-Goal Interdependency Graph). En este trabajo inicial no nos centramos en este bloque, pero lo consideramos como un trabajo futuro inmediato. A continuación explicamos muy brevemente los tipos de catálogos respectivos.

2.2. Catalogando el conocimiento de diseño

Un aspecto importante del marco propuesto por Lawrence *et al.* [3], es poder organizar el conocimiento acumulado de previas experiencias para el diseño de RNFs a través de catálogos. Hay tres tipos de catálogos (Vea figura 1):

- Catálogo para tipo de RNFs: comprende los tipos particulares de RNFs, tales como seguridad, exactitud, transportabilidad, etc.
- Catálogo de métodos: comprende el conocimiento que ayuda a refinar el grafo por descomposición de metas difusas hasta llegar a la operacionalización de las mismas.
- Catálogo de reglas de correlación: comprende el conocimiento que ayuda a detectar implícitamente relaciones entre metas difusas, denominadas como correlaciones.

2.3. Evaluación

Se inicia con el grafo etiquetado a nivel de nodos hoja que vienen a ser las alternativas de diseño o implementación (operacionalizaciones). El procedimiento de evaluación es en sentido "bottom-up", es decir, los valores etiquetados se propagan de hijos a padres teniendo en cuenta las contribuciones y las correlaciones definidas. Este proceso de evaluación consta de dos pasos: Primero, calcular el impacto individual de cada interdependencia sobre las metas difusas del grafo y en segundo lugar, estos impactos individuales de todas las interdependencias son combinadas en un simple valor.

El grado de satisfacción de los nodos superiores, pueden ser inferidos de manera automática o semiautomática mediante una definición de reglas. En la siguiente sección, se presenta una descripción general de los Sistemas de Reescritura, concepto que utilizamos para la definición de un conjunto de reglas.

3. Conceptos Generales de Sistemas de Reescritura

Las ecuaciones, es decir, expresiones de la forma $t=s$ se utilizan frecuentemente para expresar propiedades o formular definiciones de objetos. Son muy comunes en matemáticas, lógica, inteligencia artificial y programación [12].

Cuando empleamos ecuaciones para definir funciones, resulta natural *orientar la ecuación* en el sentido en el que realizamos la definición. En este caso, en lugar de ecuaciones tenemos reglas de reescritura. En la tabla 1, se muestra un ejemplo donde se define la función "append" bajo un sistema de ecuaciones y reglas de reescritura.

Tabla 1. Definición de la función append

Ecuaciones	Reglas de Reescritura
append([ ],x) = x append(x:y,z)=x:append(y,z)	Append([ ],x) → x Append(x:y,z)→x:append(y,z)

Por lo tanto, sea T un conjunto de todos los términos sobre algún vocabulario. Un sistema de reescritura sobre T es un conjunto de reglas de la forma $l \rightarrow r$, donde l y r son términos que contienen funciones y variables que pertenecen al conjunto T [6]. Dado:

$l \rightarrow r$, *donde:*
$l, r \in T(F, X)$ *F representa al conjunto de funciones y X representa al conjunto de variables*
$l \notin X$
$Var(r) \subseteq Var(l)$

El mecanismo computacional es la **reescritura de términos**, es decir, el reemplazo dentro de un término de instancias de la parte izquierda de una regla por la correspondiente instancia de la parte derecha. Ejemplo:

Para el término **append(a:b:[], c:[])** utilizando el sistema de reescritura siguiente:

R1: append([], x) → x

R2: append(x:y, z) → x:append(y, z)

Tenemos :

append(a:b:[], c:[])→ a:append(b:[], c:[]) …por *R2*
→ a:b:append([], c:[]) …por *R2*
→ a:b:c:[] ....….....por *R1*

Finalmente, en este trabajo la teoría de los sistemas de reescritura la empleamos para implementar estrategias de soporte a la toma de decisiones.

4. Proceso de Evaluación de Requisitos No Funcionales con Sistemas de Reescritura

Como se observa en la figura 1, el proceso de evaluación consta de dos pasos: La transformación del grafo a términos de reescritura, mediante la definición de funciones. Y el proceso de evaluación que consiste en inferir en base al conjunto de reglas definidas mediante la reescritura de términos. Cada uno de estos pasos se explica a continuación.

4.1. Transformación a Términos de Reescritura

Para la transformación del grafo (representación notacional) a términos de reescritura, en primer lugar, como se indicó en la sección anterior identificamos el conjunto de términos de reescritura comprendido por el conjunto de funciones y variables: *T(F,X)*. En nuestro caso, los atributos definidos tanto para las metas difusas como operacionalizaciones, son identificadas como funciones de aridad nula llamadas también **constantes.** Los diferentes tipos de correlaciones y contribuciones de tipo Make, Help, Break, Hurt, Equal, Unknown son identificadas como **funciones** de aridad uno. Y los tipos de contribuciones AND y OR son identificadas como funciones de aridad dos.

Además de estas funciones, para el cálculo del impacto global mediante la combinación de todos los impactos individuales determinados por las contribuciones explícitas e implícitas, definimos también la función "Combination" de aridad dos. (Vea Tabla 2)

Tabla 2. Identificación de elementos para transformar a términos

Elementos del SIG	Términos
1. Atributos de elementos intencionales: denegado, débilmente denegado, indefinido, débilmente satisfecho, satisfecho, conflicto.	**1. Constantes:** $F_0 = \{D, W-, U, W+, S, C\}$
2. Contribuciones: • And, Or. • Make, Help, Break, Hurt, Equal, Unknown.	**2. Funciones:** $F_1 = \{Make, Help, Break, Hurt, Equal, Unknown\}$ *(aridad uno)* $F_2=\{And,Or,Combination^3\}$ *(aridad dos)* $F = F_0 \cup F_1 \cup F_2$

En segundo lugar, establecemos un orden de precedencia para las constantes en función al grado de satisfacción definido en la sección 2.1:

$$D < W- < C < U < W+ < S$$

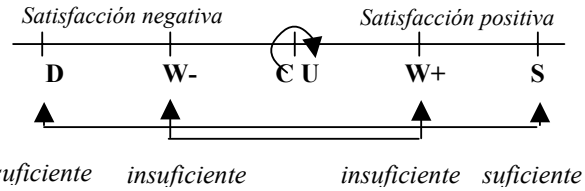
Este orden de precedencia, facilita la definición de las contribuciones AND y OR en base a los operadores difusos: "min" y "max".

Las funciones F_1 y F_2 son definidas a través de reglas de reescritura las cuales son explicadas a continuación.

4.2. Definición de Reglas de Reescritura.

En esta sección, presentamos el conjunto de reglas que definen las funciones identificadas anteriormente agrupadas según su aridad.

4.2.1. Funciones de aridad unaria: Si tenemos en cuenta las definiciones descritas en el punto 2.1.4 acerca de los tipos de contribución para la evaluación del impacto individual establecemos las siguientes relaciones entre constantes o grados de satisfacción.



[3] "Combination" no es ningún tipo de contribución, pero es una función definida para cubrir el impacto global del proceso de evaluación

De esta manera, por ejemplo: el opuesto de una satisfacción negativa suficiente es un tipo de satisfacción positiva suficiente, siendo una relación bidireccional. Para el caso de indefinición o de conflicto la relación es consigo misma independiente del tipo de contribución (Fila). Además se llegará también al estado de indefinición cuando tenemos una contribución desconocida (Columna).

El impacto individual de las contribuciones sobre los diferentes grados de satisfacción es calculado como se ve en la siguiente tabla.

Tabla 3. Posibles combinaciones para evaluar un impacto individual.

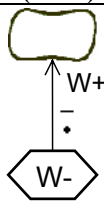
Impacto Individual	Tipo de Contribución					
	Break	**Hurt**	**Un-known**	**Help**	**Make**	**Equal**
D	W+	W+	U	W-	W-	D
W-	W+	W+	U	W-	W-	W-
C	C	C	U	C	C	C
U	U	U	U	U	U	U
W+	W-	W-	U	W+	W+	W+
S	D	W-	U	W+	S	S

Por consiguiente tenemos las siguientes reglas de reescritura de la forma $l \rightarrow r$. Donde r siempre es una constante.

Tabla 4. Reglas de reescritura para funciones de aridad unaria.

R1: Make(D) → W-	*R13: Break(D) → W+*
R2: Make(W-) → W-	*R14: Break(W-) → W+*
R3: Make(C) → C	*R15: Break(C) → C*
R4: Make(U) → U	*R16: Break(U) → U*
R5: Make(W+) → W+	*R17: Break(W+) → W-*
R6: Make(S) → S	*R18: Break(S) → D*
R7: Help(D) → W-	*R19: Hurt(D) → W+*
R8: Help(W-) → W-	*R20: Hurt(W-) → W+*
R9: Help(C) → C	*R21: Hurt(C) → C*
R10: Help(U) → U	*R22: Hurt(U) → U*
R11: Help(W+) → W+	*R23: Hurt(W+) → W-*
R12: Help(S) → W+	*R24: Hurt(S) → W-*
R25: Equal(x) →x, $\forall x \in F_0$	
R26: Unknown(x) →U, $\forall x \in F_0$	

Ejemplo:

Representación Notacional (GRL)	Regla de Reescritura
	*Hurt(W-) →W+*

4.2.2. Funciones de aridad binaria: Las funciones F_2, relacionan un grupo de nodos hijos a un nodo padre. Cabe indicar, que las propiedades que se exigen para este sistema de reescritura, son tanto la conmutatividad como la asociatividad:

Sea: $\forall x, y, z, w \in F_0 \wedge \forall f \in F_2$

Conmutativa: *f(x,y) → z = f(y,x) → z*
Asociativa: *f (x, f (y,z)) → w= f (f(x,y),z) → w*

La regla de reescritura que define la función AND según el orden de precedencia, se obtiene de calcular el **valor mínimo** de las constantes que componen dicha función.

R27: AND(x,y) →min(x,y)

La regla de reescritura que define la función OR según el orden de precedencia, se obtiene de calcular el **valor máximo** de las constantes que componen dicha función.

R28: OR (x,y) →max(x,y)

Ejemplo:

AND (D,U) → D
AND(W-,W+) → W-
OR (D,U) → U
OR (W-,W+) → W+

Antes de definir el conjunto de reglas para la función "Combination", explicamos con un ejemplo la necesidad de expresar esta función. En el siguiente gráfico, se ve un nodo padre influenciado por dos tipos de contribución MAKE y BREAK. Luego de una evaluación individual para cada nodo es necesario combinar los valores obtenidos S y D en un solo valor, obteniéndose para este caso un CONFLICTO (C) por tratarse de valores opuestos y por lo tanto una contradicción.

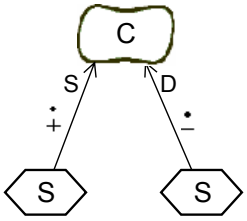


Figura 2. Ejemplo de evaluación: Impacto Global

Para definir el conjunto de reglas, previamente se analizó todas las posibles combinaciones con las seis constantes definidas, llegando a las siguientes conclusiones:

- Para las combinaciones que tienen en alguno de sus parámetros de entrada a la constante Conflicto (C), tendrá como consecuente la constante Conflicto (C).

  R29: Combination(x,C) → *C* , $\forall x \in F_0$

- Las combinaciones que presentan en alguno de sus parámetros la constante Indefinido (U), se mantiene como consecuente el mismo valor indefinido.

  R30: Combination(x,U) → *U* , $\forall x \in F_0$

  Ejemplo:

  Combination(D,U) → *U*
  Combination(U,W+) → *U*

- Las combinaciones que tienen como parámetros opuestos, se obtiene como consecuente la constante Conflicto (C). Además el proceso de evaluación para ese nodo ya no prosigue.

  R31: Combination(x,y) → *C* , $\forall x, y \in F_0 \quad |x = \neg y$

  Ejemplo: Siendo los dos únicas reglas identificadas:

  Combination(D,S) → *C*
  Combination(W-,W+) → *C*

- Las combinaciones restantes se definirán de la siguiente manera:

  R32: Combination(D,D) → *D*
  R33: Combination(D,W-) → *D*
  R34: Combination(D,W+) → *D*
  R35: Combination(W-,W-) → *D*
  R36: Combination(W+,W+) → *S*
  R37: Combination(S,W-) → *S*
  R38: Combination(S,W+) → *S*
  R39: Combination(S,S) → *S*

5. Ejemplo Práctico: Facilidad de Aprendizaje.

Con la finalidad de hacer una verificación inicial del sistema de reescritura definido en el presente trabajo, se ha construido un pequeño ejemplo, resultado **parcial** de la evaluación de usabilidad de un prototipo de herramienta CASE[1]. Se evaluó la interfaz de esta herramienta en base a los criterios ergonómicos desarrollados por Bastien y Scapin[2]. Partimos de un modelo estándar, las subcaracterísticas de usabilidad definidas en el modelo de calidad ISO 9126[9], y a partir de cada una de estas subcaracterísticas procedemos con un refinamiento para llegar a un nivel de atributos y alternativas de diseño (operacionalizaciones). La figura 3, representa el grafo de metas difusas e interdependencias para el requisito no funcional *Facilidad de Aprendizaje*, que contribuirá de manera positiva a la usabilidad [9].

Como la evaluación del grafo es un sentido bottom-up, la idea es ir etiquetando en función a los valores de entrada al nivel de hojas del grafo, definidos por el diseñador. A continuación explicamos paso a paso este proceso de evaluación.

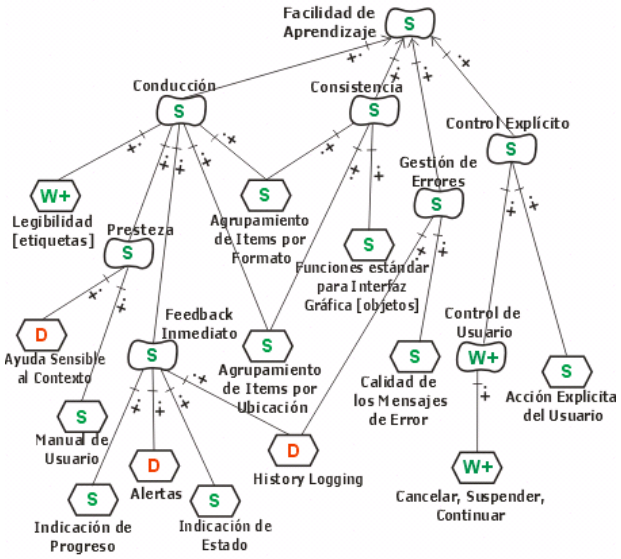


Figura 3. Grafo de Metas Difusas e Interdependencias Parcial

En el grafo se observan 8 metas difusas que deben ser etiquetadas: *Presteza, Feedback Inmediato,*

Consistencia, Gestión de Errores, Control de Usuario, Conducción, Control Explícito, y Facilidad de Aprendizaje.

La transformación a términos de reescritura para la meta difusa "***Presteza"*** sería:
Combination(Make(D), Make(S))

El mecanismo de reescritura para la obtención de su valor sería:
*Combination**(Make**(D),**Make**(S))*→*Combination**(Make**(D),**S)*…. ***.....R6***
→*Combination**(W-,S)*……..***R1***
→*S*…….. ***…...R37***

La meta difusa "***Feedback Inmediato***", la expresamos de la siguiente forma:
*Combination**(Make**(S), **Make**(D), **Make**(S), **Make**(D))*

El mecanismo de reescritura para la obtención de su valor sería:
Combination(Make(S), Make(D), Make(S), Make(D)) →
Combination(S,W-,S,W-)……R1, R6
→ *S……..R37, R39*

La meta difusa "***Consistencia***", la expresamos de la siguiente forma:
*Combination**(Make**(S), **Make**(S), **Make**(S))*

El mecanismo de reescritura para la obtención de su valor sería:
Combination(Make(S), Make(S), Make(S)) →*Combination(S,S,S) ……R6*
→*S ……R39*

La meta difusa "***Gestión de Errores***", la expresamos de la siguiente forma:
*Combination**(Help**(D), **Make**(S)))*

El mecanismo de reescritura para la obtención de su valor sería:
*Combination**(Help**(D), **Make**(S))* →*Combination(W-,S)….R6, R7*
→*S……R37*

La meta difusa "***Control de Usuario***", la expresamos de la siguiente forma:
Make(W+)

Siendo el mecanismo de reescritura muy sencillo:
Make(W+) → *W+……R5*

La meta difusa "***Conducción"***:
Combination(Make(W+),Make(S),Make(S),Make(S),Make(S))
El mecanismo de reescritura para la obtención de su valor sería:
Combination(Make(W+),Make(S),Make(S),Make(S),Make(S))
→*Combination(W+,S,S,S,S)….R5,R6*
→ *S……R38,R39*

La meta difusa "***Control Explícito",*** la expresamos de la siguiente forma:
Combination(Make(W+),Make(S))

El mecanismo de reescritura para la obtención de su valor sería:
Combination(Make(W+),Make(S)) → *Combination(W+,S)...R5, R6*
→ *S……R38*

La meta difusa "***Facilidad de Aprendizaje",*** la expresamos de la siguiente forma*:*
And(Make(S), Make(S), Make(S), Make(S))

El mecanismo de reescritura para la obtención de su valor sería:
And(Make(S),Make(S),Make(S),Make(S)) →*And(S,S,S,S)…..R6*
→ ***S****……R27*

Eso significa, que la subcaracterística de Usabilidad "Facilidad de Aprendizaje" es satisfecha, con este ejemplo hemos podido ver como las submetas difusas contribuyen a las metas difusas padre y que a pesar de que algunas alternativas de diseño fueron denegadas, esta subcaracterística resulta siendo satisfecha.

La herramienta NFR-Assistant no permite al usuario asignar los tipos de satisfacción: Débilmente Satisfecho y Débilmente Denegado; opciones que si están presentes en nuestra arquitectura por considerarlas sumamente necesarias, ya que no siempre se tendrá en el grafo un gran nivel de detalle como se observa en este ejemplo.

El seguimiento realizado resulta tedioso si lo hacemos manualmente, por tal razón se ha implementado una aplicación con la finalidad de semi-automatizar este proceso de evaluación en base al sistema de reescritura definido. Se ha utilizado Visual Prolog para dicha implementación, el cual por ser un lenguaje lógico declarativo nos facilita la representación del conocimiento mediante reglas, en este caso reglas de reescritura.

6. Conclusiones y Trabajos Futuros

En este trabajo, se ha descrito un proceso de evaluación de impacto de decisiones, el cual consiste en evaluar las metas identificadas y sus interacciones. Este proceso puede llevarse a cabo de manera semi-automática mediante la implementación de un conjunto de reglas que han sido expresadas en términos de reescritura, lo cual contribuyó a la expresividad y rigurosidad de la definición de estas reglas.

Además, se ha descrito un esquema general que representa los elementos necesarios para la gestión de RNFs. En cuanto a su representación notacional se utilizó el lenguaje de requisitos orientado a metas GRL [8]. Finalmente, con el fin de ilustrar el proceso de evaluación y verificar las reglas propuestas, se ha construido un grafo de metas difusas para evaluar el requisito no funcional facilidad de aprendizaje de los usuarios al utilizar una herramienta de gestión de RNFs propuesto por Antani [1].

Como trabajo futuro se tiene planeado aplicar el proceso de evaluación a más casos de estudio para validar las reglas propuestas.

Agradecimientos

Los autores agradecen al profesor Dr. Salvador Lucas, por sus comentarios y sugerencias oportunos relacionado a sistemas de reescritura. Así mismo al Dr. Oscar Pastor por todas sus observaciones y sugerencias de mejora.

Referencias

[1] S. Antani, "A Collaborative Case Tool for the Non-Functional Requirements Framework" M.S. Thesis, Dept. of Computer Science, Univ. of Texas Dallas, 2006.

[2] Bastien, J. M. and Scapin, D. L. "Ergonomic Criteria for the Evaluation of Human-Computer Interfaces", version 2.1, 1993.

[3] L. Chung, B. Nixon, E. Yu and J. Mylopoulos, *Non Functional Requirements in Software Engineering*, *Kluwer Academic Publishers*, London, 2000.

[4] L.M. Cysneiros, J.C.S.P. Leite, and J. Neto, "A Framework for Integrating Non-Functional Requirements into Conceptual Models", *Requirements Engineering Journal-Vol 6-Issue 2*, Apr. 2001, pp 97-115.

[5] L.M. Cysneiros and J.C.S.P. Leite., "Using UML to Reflect Non-Functional Requirements", *Proceedings of 11th CASCON*, IBM, Toronto-Canada, Nov. 2001, pp 202-216.

[6] N. Dershowitz, Examples of Termination, University of Illinois, USA.

[7] X. Franch, "Systematic Formulation of Non-Functional Characteristics of Software", *Proceedings of 3rd International Conference on Requirements Engineering (ICRE)*, IEEE Computer Society, Colorado Springs-USA, 1998.

[8] GRL Ontology, http://www.cs.toronto.edu/km/GRL/, Acceso en Octubre 2006.

[9] ISO/IEC 9126-1:2001, Software engineering - Product quality - Part 1: Quality model.

[10] J. Lee, "Extending the Potts and Bruns Model for Recording Design Rationale", 13th Intl. Conj on Software Engineering, IEEE-ACM, 1991, pp 114-125.

[11] J.C.S.P. Leite and A.P.M. Franco, "A Strategy for Conceptual Model Acquisition", Proceedings of 1st IEEE International Symposium on Requirements Engineering, IEEE Computer Society Press, San Diego(Ca)-USA, 1993, pp 243-246.

[12] S. Lucas, Apuntes de Asignatura de Doctorado.

[13] J. Mylopoulos, L. Chung and B. Nixon, "Representing and Using Non-Functional Requirements: A Process-Oriented Approach", IEEE Trans. on Software Engineering, Jun. 1992, pp 483-497.

[14] A. Ruiz, "Una Aproximación Semicualitativa al tratamiento Automático de Requisitos de Calidad", Tesis Doctoral, Sevilla-España, May. 2002.

[15] G. Salazar-Zárate and P. Botella., "Use of UML for Modeling Non-Functional Aspects", Proceedings of 13th International Conference Software & Systems Engineering and their Applications(ICSSEA'2000), Paris-France, 2000.

[16] OME3, http://www.cs.toronto.edu/km/ome/index.html, Acceso en Octubre 2006.

Atributos Contextuales Relevantes para la Selección de Técnicas de Educción de Requisitos

Dante Carrizo
Departamento Sistemas Informáticos y Computación
Facultad de Informática
Universidad Complutense de Madrid
C/ Prof. José García Santesmases s/n, 28040, Madrid, España
dcarrizo@fdi.ucm.es

Oscar Dieste
Departamento de Lenguajes y Sistemas Informáticos e Ingeniería de Software
Facultad de Informática
Universidad Politécnica de Madrid
Avda. Montepríncipe s/n, 28040, Boadilla del Monte, 28660, España
Email: odieste@fdi.ucm.es

Abstract

La Ingeniería de Requisitos puede hacer uso de una gran cantidad de técnicas para educir las necesidades de los usuarios. No obstante, no existe una guía práctica completa para realizar la selección de las técnicas más adecuadas en un proyecto de desarrollo particular. Para poder generar un marco que apoye esta decisión se debe determinar los atributos del contexto que influyen en la diferenciación de efectividad de las técnicas. Este artículo presenta un estudio que arroja una propuesta de un conjunto básico de estos atributos contextuales.

1. Introducción

La ingeniería de requisitos es el proceso sistemático de desarrollar requerimientos a través de un proceso iterativo y cooperativo de análisis del problema, documentando las observaciones resultantes en una variedad de formatos de representación, y chequeando la precisión del entendimiento ganado [1]. Esta etapa comprende actividades de Educción (captura, descubrimiento, adquisición), Análisis (y negociación), Especificación, y Validación de Requerimientos. Además, establece una actividad de Gestión de Requerimientos para manejar los cambios, mantenimiento y trazabilidad de los requerimientos [2].

El proceso de educción, cuya finalidad es llevar a la luz los requisitos, no sólo es un proceso técnico para construir un sistema, sino también un proceso social que envuelve a diferentes personas, lo que conlleva dificultades añadidas a su realización [3].

Un somero análisis del campo de la Ingeniería de Requisitos permite observar claramente que la mayoría de los trabajos realizados se refieren a métodos o técnicas de representación o modelado de requisitos [4], pero no a la forma en que se obtienen esos requisitos. A pesar de que existe una gran cantidad de técnicas de educción [5], muchas de las cuáles han sido inicialmente utilizadas en la construcción de sistemas basados en conocimientos, en la mayoría de los casos se utilizan simplemente las entrevistas para la captura de información. Sin embargo, existen claras evidencias que las entrevistas tradicionales, así como otras técnicas de inspección, son inadecuadas para el desarrollo de sistemas actuales [6].

Adicionalmente, se hace evidente la falta de herramientas que soporte específicamente la actividad de educción [4] [5], así como la práctica inexistencia de una guía que permita auscultar los parámetros involucrados en el proceso y, a partir de su análisis, obtener un plan de educción adecuado que permita obtener óptimamente los requisitos [6].

En esta dirección, para poder generar una guía que recomiende las técnicas a aplicar en un caso particular, este trabajo pretende determinar un conjunto de atributos del contexto del proceso que influyen en la efectividad de las técnicas de educción.

En particular, este artículo presenta la conformación de un esquema de atributos a partir de un conjunto inicial generado en base a las propuestas de modelos relacionados, casos de estudios y experimentos.

En la sección 2, se describe el problema de la selección de técnicas adecuadas en una situación particular. En la sección 3, se analizan los trabajos previos relacionados con el problema. A continuación, en la sección 4, se expone el marco propuesto como aproximación a la solución del problema. Finalmente en la sección 5, se revisan los diferentes atributos que pueden tener influencia en la "bondad" de las técnicas.

2. Trabajos Relacionados

No existen muchos estudios previos con el fin de determinar un conjunto de atributos que influyen en la selección de técnicas de educción. A continuación se comentan brevemente aquellos que hacen una propuesta en este sentido.

Un estudio presentado por Byrd y otros [7] presenta un esquema de categorización basado en tres dimensiones: obstáculos de comunicación (de los participantes), entendimiento del dominio del problema (información, proceso, comportamiento, marco problema), y control del proceso (conducido por el Usuario/Experto, conducido por el Analista/Ingeniero de Conocimiento, conducido por la Máquina/Sistema de apoyo). Claramente la cantidad de atributos es muy reducida pero es un serio intento por recomendar el uso de un número importante de técnicas en esos casos.

Maiden y Rugg [8] proponen un marco denominado ACRE (Acquisition Requirements), para asistir a los Ingenieros de Requisitos en la selección de métodos o técnicas para la adquisición de requisitos. Su objetivo es proveer una guía para seleccionar de entre una docena de técnicas o métodos con diferentes características y diferentes orígenes. Para realizar la elección de la (las) técnica (s) apropiada (s), el estudio considera cinco facetas o atributos: Propósito de los Requisitos, Tipos de conocimientos, Filtro interno de conocimiento, Fenómenos Observables, Contexto de Adquisición. También es un enfoque reducido en cuanto a la cantidad de facetas que intervienen en la decisión, aunque el esquema de uso de un grupo de técnicas es práctico.

Dhaliwal y Benbazat [9] presentan un marco en que proponen variables moderadoras de aspectos como: Características del experto, Características del ingeniero, grado de incertidumbre, Tipo de Tarea, Métodos de resolución de problemas, Metodología de desarrollo, y Entorno organizacional. Este marco es sólo descriptivo, presentando los atributos sin mayor detalle de definición y sin prescripción de las técnicas.

Finalmente, Davis y Hickey [10] proponen una ontología de las características de la situación contextual en que las técnicas se aplican. En esta propuesta se definen cinco categorías de atributos: Características del dominio del problema (tales como complejidad, borrosidad, madurez, importancia de requisitos no funcionales); Características del dominio de solución (tipo de solución a desarrollar); Características de los interesados; Características de los desarrolladores (tales como el conocimiento, destrezas comunicacionales, experiencia); y Características de los ingenieros de requisitos (conocimiento del dominio del problema o solución por ejemplo). Este es el trabajo más completo en cuanto a atributos a considerar aunque su definición en detalle no ha sido entregada formalmente.

3. Marco de Investigación

La generación del conjunto de atributos se realiza a partir del análisis objetivo y teórico de un conjunto inicial obtenido de las propuestas encontradas en los artículos científicos previos.

Para poder analizar el esquema inicial de atributos, se establecen algunos criterios que ayudarán en la decisión sobre cada atributo. Estos criterios se relacionan con la viabilidad de conformar completamente el esquema, es decir, los atributos con sus valores, para que el marco sea fácilmente utilizable por los usuarios finales (ingenieros de requisitos), y con la certeza de que el atributo, al menos en algún grado, diferencia efectividades entre las técnicas.

Estos criterios son:

- (FV) Facilidad de Valoración: Posibilidad de distinguir opciones de valores distintos para el atributo. (B) Baja, (M) Media, (A) Alta.
- (FI) Facilidad de Instrumentación: Posibilidad de dar un valor determinado al atributo en un proyecto a desarrollar. (B) Baja, (M) Media, (A) Alta.
- (JT) Justificación Teórica: Posibilidad de encontrar justificación de su influencia. (N) No hay, (P) Posiblemente, (S) Sí hay.

Después de aplicar los criterios se decide la acción a tomar con cada atributo. Las acciones que se pueden tomar con respecto a los atributos son:

- (A) Aceptar: Se acepta tal cual aparece propuesto.
- (E) Eliminar: Se elimina por no cumplir criterios de elegibilidad.
- (R) Reunir: Se funde con otro atributo por su similitud.
- (M) Modificar: Se modifica su Nombre para darle el sentido correcto.
- (+) Agregado: Sumado a la lista por los investigadores.
- (?) Pendiente: Atributo poco relevante. Postergado para análisis posterior.

En particular, para que un atributo sea eliminado debe presentar BAJA Facilidad de Valoración, ó BAJA Facilidad de Instrumentación ó NO HABER Justificación de Influencia. Este requisito se debe a que si se desea que el marco propuesto sea realmente una guía práctica debiera ser posible construir una herramienta software que lo soporte. Para ello, los valores de los atributos deben ser claramente distinguibles y se debe poder asignar un valor, en un

caso particular de desarrollo, con cierta facilidad y rapidez.

4. Determinación del Esquema de Atributos

4.1. Esquema inicial

El Esquema inicial se obtiene de la investigación relacionada con las técnicas de educción y de la literatura asociada. Algunos atributos provienen de la experiencia de autores que proponen marcos de solución al proceso de selección de técnicas, y la mayoría, de estudios empíricos y casos de estudio que intentan demostrar su influencia en la diferencia de resultados de la aplicación de las técnicas.

En la Tabla 1 se presenta la totalidad de los atributos agrupados en los factores antes descritos. Para cada uno de ellos, se muestran los autores que los tratan en sus estudios, describiendo los valores que pueden tomar y la naturaleza y resultado del estudio.

4.2. Análisis de los atributos

A partir del Esquema Inicial se debe obtener el conjunto de atributos definitivo para conformar el Esquema Final. Para ello, se analiza el Esquema inicial justificando para cada caso su permanencia, modificación o eliminación según los criterios definidos en la sección 3. Además, los investigadores han agregado algunos atributos que, fundamentado en su experiencia, estiman que influyen en la efectividad de las técnicas.

4.3. Esquema final de atributos

Producto del análisis de los atributos iniciales se genera un esquema final de atributos que se presentan en la Tabla 2. En ella, se define brevemente cada atributo y los valores propuestos para ellos.

5. Conclusiones

El objetivo de este trabajo pretendía generar un esquema de atributos básicos y relevantes para las técnicas de educción de requisitos. De los 29 atributos iniciales, se han seleccionado 12, ya sean aceptados tal cual, modificados o reunidos en otro, y 4 atributos nuevos agregados por el investigador.

A los atributos elegidos se le han definido valores distinguibles y excluyentes que pueden asistir en la recomendación del uso de una técnica particular, para un proyecto real de desarrollo. El siguiente paso en la presente investigación, es la generación de la guía de adecuación de las diferentes técnicas de educción disponibles para cada uno de los valores de los atributos determinados.

6. Referencias

[1] Loucopoulos, P., and V. Karakostas. Systems Requirements Engineering. McGraw-Hill, pp. 1995.

[2] Kotonya, G., and I. Sommerville. Requirements Engineering: Processes and Techniques. John Wiley and Sons, 1998.

[3] Chatzoglou P., Soteriou A. "A DEA framework to assess the efficiency of the software requirements capture and analysis process". Decision-Sciences. vol.30, no.2; Spring 1999; p.503-31. 1999.

[4] Kovitz, B. Practical Software Requirements. Manning Publications Co. 1998.

[5] Goguen, J., and C. Linde. "Techniques for Requirements Elicitation". International Symposium on Requirements Engineering, Los Alamitos, California: IEEE Computer Society Press, January 1993, pp. 152-164.

[6] Holtzblatt, K., and H. Beyer. "Requirements Gathering: The Human Factor". Communications of the ACM, 38, 5 (May 1995), pp. 31-32.

[7] Byrd, T.A., Cossick, K.L. and Zmud, R.W. "A Synthesis of Research on Requirements Analysis and Knowledge Acquisition Techniques". MIS Quarterly, 16, 117-138. 1992.

[8] Maiden, N., and G. Rugg. "Knowledge Acquisition Techniques in Requirements Engineering". In Proc. Of the Workshop on Requirements Elicitation for Systems Specification, Keele, UK, July 1994.

[9] Dhaliwal Jasbir Singh and Izak Benbazat. "A framework for the comparative evaluation of knowledge acquisition tools and techniques". Knowledge-Acquisition. vol.2, no.2; June; p.145-166. 1990.

[10] Davis, A., and A. Hickey. "Requirements Elicitation and Requirements Elicitation Technique Selection: A Model of Two Knowledge-Intensive Software Development Processes". Proceedings of the Thirty-Sixth Hawaii International Conference on System Sciences, Los Alamitos, California: IEEE Computer Society Press, 2003.

[11] Lloyd, W.J. "Tools and Methods for effective distributed requirements engineering: an empirical study". Master Thesis. Virginia Tech, 2001. Available at: http://scholar.lib.vt.edu/theses/available/etd-07262001-110924/ 2001.

[12] Agarwal R. and Tanniru M. “Knowledge Acquisition using Structured Interviewing: an Empirical Investigation”. Journal of Management Information Systems, 7, 1. Summer 1990, 123-140. 1990. El Emam, K., and N. Madhavji. “Requirements Engineering Practices in Information Systems Development: A Multiple Case Study”. Second International Symposium on Requirements Engineering, Los Alamitos, California: IEEE Computer Society Press, 1995.

[13] Roth Roberta M. and William C. Wood II. “Knowledge Acquisition from single versus multiple experts: a field study comparison using the Delphi technique”. The journal of Knowledge Engineering. Volume 6, Number 3. Fall. 1993.

[14] Burton-AM; Shadbolt-NR; Rugg-G; Hedgecock-AP. “The efficacy of knowledge elicitation techniques: a comparison across domains and levels of expertise”. Knowledge-Acquisition. vol.2, no.2; June; p.167-78. 1990.

[15] Chao-C-J; Salvendy-G. “Impact of cognitive abilities of experts on the effectiveness of elicited knowledge”. Behaviour and Information Technology. vol.14, no.3; May-June 1995; p.174-82. 1995.

[16] Browne G. J., Rogich M. B. “An empirical investigation of user requirements elicitation: comparing the effectiveness of prompting techniques”. Journal of Management Information Systems. vol.17, no.4, Spring 2001. p.223-49. 2001.

[17] McCloskey-BP; Geiwitz-J; Kornell-J. “Empirical comparisons of knowledge acquisition techniques”. Proceedings of the Human Factors Society 35th Annual Meeting. Human Factors Soc, Santa Monica, CA, USA; 2 vol. xxiii+1631 pp. p.268-72 vol.1. 1991.

[18] Kim J. and J. Courtney. “A survey of knowledge acquisition techniques and their relevance to managerial problem domains”. Decision Support Systems, 4, 269-284. 1988.

[19] Grabowski M. “Knowledge acquisition methodologies: survey and empirical assessment”. In proceedings of the Ninth International Conference on Information Systems, pages 47-54. 1988.

[20] Shadbolt, N. R and Burton, A. M. “Empirical Studies in Knowledge Elicitation”. ACM-SIGART Special Issue on Knowledge Acquisition 108. 1989.

[21] Fazlollahi, B. and Tanniru, M. R. "Selecting a requirements determination methodology-contingency approach revisited". Information and Management, 21:5, pp. 291-303. 1991.

[22] W.P. Wagner, Q.B. Chung & M.K. Najdawi. “The impact of problem domains and knowledge acquisitions techniques: a content analysis of P/OM ES case studies”. Expert Systems with Applications, 24, p.79-86. 2003.

[23] Holsapple C. and V. Raj. “An exploratory study of two KA methods”. Expert Systems, 11 (2), 77-87.1994.

[24] Keil M. and Carmel E. “Customer-Developer Links”. Communications of the ACM, 38, 5 (May 1995), pp. 33-44.

Tabla 1. Esquema inicial y análisis de atributos.

Factor	Atributos	FV	FI	JT	Acción	Autores	Observación
Eductor	Experiencia en Ingeniería de Requisitos (Educción)	A	A	S	M	Lloyd 2002 [11]	Débil relación medida. Se contabiliza cada participación pasada en actividad de IR.
						Agarwal and Tanniru 1990 [12]	No hubo evidencia de influencia entre las técnicas de entrevistas para novicios y expertos.
						Dhaliwal y Benbazat 1990 [9]	Propuesta de los autores. No presenta evidencia empírica.
	Conocimiento Técnico de Métodos de Educción	A	A	S	M	Dhaliwal y Benbazat 1990[9]	Propuesta de los autores. No presenta evidencia empírica. El entrenamiento en métodos facilita el uso de ciertas técnicas.
	Conocimiento (Familiaridad) del Dominio	A	A	S	M	Dhaliwal y Benbazat 1990[9]	Propuesta de los autores. No presenta evidencia empírica. Un mayor conocimiento del dominio facilita el uso de ciertas técnicas.
	Experiencia en (la Técnica)Métodos de Educción	A	A	S	M	Dhaliwal y Benbazat 1990[9]	Propuesta de los autores. No presenta evidencia empírica. Modificado para cada técnica en particular.
	Problemas Cognitivos	M	M	P	?	Byrd, Cossick and Zmud 1992 [7]	Opinión de autores, muy general. Poco relevante en el eductor.
Educente	Número de Usuarios	A	A	S	A	Maiden y Rugg 1996 [8]	Opinión de los autores. La cantidad de sujetos pre-condiciona la aplicación de ciertas técnicas.
	Número de Expertos	A	A	S	R	Roth y Wood 1993 [13]	Evidencia empírica de diferencia de eficacia.
	Participación del Usuario	A	M	S	M	Lloyd 2002 [11]	Débil relación medida. Valoración subjetiva de la participación de los usuarios en la sesión. Atributo reformulado.
	Localización/ Accesibilidad	A	A	P	+		La ubicación del usuario pre-condiciona la aplicación de ciertas técnicas.
	Disponibilidad Tiempo	A	A	P	+		Se agrega pues solo si el usuario tiene tiempo pueden aplicarse algunas técnicas.
	Nivel de Pericia	A	A	S	A	Burton y otros 1990 [14]	Leve diferencia entre técnicas comparadas.
						Dhaliwal y Benbazat 1990 [9]	Propuesta de los autores. No presenta evidencia empírica.
	Estilos Cognitivos (Capacidad de Articulación)	M	M	P	M	Dhaliwal y Benbazat 1990 [9]	Propuesta de los autores. No presenta evidencia empírica. Atributo reformulado.
	Variables de Personalidad	B	B	P	E	Dhaliwal y Benbazat 1990 [9]	Propuesta de los autores. No presenta evidencia empírica. Difícil valoración e instrumentación.
	Problemas Cognitivos (Consenso entre Fuentes)	M	M	S	M	Byrd, Cossick y Zmud 1992 [7]	Opinión de autores, muy general. Atributo reformulado.
	Habilidades Cognitivas	M	B	P	¿	Chao y Salvendy 1995 [15]	Valoración para cada habilidad (Facilidad de asociación, de expresión, de imaginación, de inducción, de razonamiento, etc.). Evidencia empírica de su influencia en la efectividad de las técnicas.
Dominio del Problema	Tipo de Fenómenos	A	A	P	?	Maiden y Rugg 1996 [8]	Visión basada en el conocimiento de los autores.
	Tipo de Información	M	M	S	M	Browne y Rogich 2001 [16]	Validación empírica y estadística de la efectividad de las técnicas para obtención de Metas, Proceso, Tarea o Información.

						Maiden y Rugg 1996 [8]	Visión basada en el conocimiento de los autores para valores de Datos, Procesos, y Comportamiento.
						McCloskey y Geiwitz 1991 [17]	Evidencia empírica de influencia para capturar conocimiento Declarativo y Procedimental.
						Kim y Courtney 1988 [18]	Visión basada en el conocimiento de los autores. No hay validación empírica en la captura de Conceptos, Heurísticas y Razonamiento.
						Maiden y Rugg 1996 [8]	Visión basada en el conocimiento de los autores. Recomendaciones para captura de Conocimiento.
	Tipo de Heurísticas	A	M	P	R	Grabowski 1988 [19]	Estudio Experimental. Validez reducida. Reunido en otro atributo.
	Campos de Dominios	B	M	P	E	Shadbolt y Burton 1989 [20]	Resultados no concluyentes. Difícil valoración.
	Grado de Estructura Percibida	M	B	N	E	Dhaliwal y Benbazat 1990 [9]	Opinión de los autores. No presenta evidencia empírica ni valores del atributo.
						Kim y Courtney 1988 [18]	Basado en opinión de los autores. No están bien definidos los valores De difícil instrumentación.
	Grado de Confusión	A	M	P	?	Fazlollahi y Tanniru 1991 [21]	Propuesta de autores basada en conocimiento y experiencia propia. Poco relevante.
	Grado de Definición	A	M	P	+		Se agrega pues una baja definición del problema puede prescribir algunas técnicas.
	Grado de Incertidumbre	A	M	P	?	Dhaliwal y Benbazat 1990 [9]	Opinión de los autores. No presenta evidencia empírica ni valores.
						Fazlollahi y Tanniru 1991 [21]	Propuesta de autores basada en conocimiento y experiencia propia. Considerado en otro atributo.
	Tipo de Tareas	A	A	S	?	Dhaliwal y Benbazat 1990 [9]	Propuesta de los autores. No presenta evidencia empírica ni valores.
						Wagner, Chung y Najdawi 2003 [22]	Medición empírica de claro impacto del tipo de problema en el uso de las técnicas.
						Chao y Salvendy 1995 [15]	Evidencia empírica de su influencia en la efectividad de las técnicas. Sólo considera los valores de Diagnóstico, Depuración e Interpretación.
	Entidades del Dominio	A	A	S	R	Byrd, Cossick y Zmud 1992 [7]	Propone diferentes tipos de información a capturar. Opinión de los autores. Se reúne en otro atributo.
	Tamaño	A	B	P	E	Kim y Courtney 1988 [18]	Basado en opinión de los autores. No están bien definidos los valores. Difícil instrumentación.
	Complejidad	A	B	S	?	Kim y Courtney 1988 [18]	Basado en opinión de los autores. No están bien definidos los valores.
						Holsapple y Raj 1994 [23]	Evidencia de que el factor influye en la selección. Difícil instrumentación.
Dominio de la Solución	Tipo de Producto	A	A	S	?	Keil y Carmel 1995 [24]	Conclusión basada en la experiencia y conocimiento de los autores y en encuesta a jefes de proyecto. Discutible relevancia.
	Métodos de Resolución	M	B	N	E	Dhaliwal y Benbazat 1990 [9]	Opinión de los autores. No presenta evidencia empírica. Difícilmente valorable.
Proceso de Educción	Propósito de los Requisitos	M	M	N	E	Maiden y Rugg 1996 [8]	Visión basada en el conocimiento de los autores. Atributo poco relevante.
	Restricciones	M	A	S	M	Maiden y Rugg 1996 [8]	Se consideran principalmente las restricciones de tiempo y recursos del proyecto.
	Momento del Proceso	M	A	P	+		Se agrega pues la etapa del proceso pre-condiciona el uso de ciertas técnicas.
	Metodología de Desarrollo	M	A	P	?	Dhaliwal y Benbazat 1990 [9]	Opinión de los autores. No presenta evidencia empírica. Poco relevante.

Tabla 2. Esquema final de atributos.

Factor	Atributos	Definición	Valor	Descripción
Eductor	Experiencia en Educción	Actividades de educción previas abordadas por el eductor	Alta	Más de 5 proyectos educción
			Media	Entre 2 y 5 proyectos de educción
			Baja	Menos de 2 proyectos de educción
	Experiencia en la técnica de Educción	Actividades previas del eductor de aplicación de la técnica	Alta	Más de 5 aplicaciones de la técnica
			Media	Entre 1 y 5 aplicaciones de la técnica
			Nula	Sin aplicación de la técnica
	Entrenamiento en Técnicas de Educción	Formación y práctica recibidas por el eductor en Técnicas de Educción	Alta	Entrenamiento formal y práctica
			Media	Entrenamiento no formal sin práctica
			Nula	No tiene conocimiento respecto de ellas
	Familiaridad con el Dominio	Proyectos previos en dominio similar o conocimiento de él, por parte del eductor	Alta	Más de 2 proyectos o conocimiento formal
			Media	Entre 1 y 2 proy o conocimiento no formal
			Baja	Completamente desconocido
Educente	Individuos en Sesión	Cantidad de individuos que pueden participar en una misma sesión de educción	Individual	1 individuo
			Grupal	Entre 2 y 5 individuos
			Masivo	Más de 5 individuos
	Interés del Educente	Motivación mostrada por el educente en participar en las sesiones	Alto	Muy interesado
			Medio	Poco interesado
			Bajo	Sin interés
	Localización/ Accesibilidad	Ubicación física del educente	Lejano	En ciudad distinta del eductor
			Cercano	En la misma ciudad del eductor
			In-situ	En el mismo edificio del eductor
	Disponibilidad de Tiempo	Tiempo del que dispone el educente para dedicar a las sesiones	Alta	Dispone holgadamente de tiempo
			Media	Dispone regularmente de tiempo
			Baja	Dispone menos tiempo del deseado
	Nivel de Pericia	Experiencia del educente en el dominio del problema o su trabajo	Experto	Más de 10 años en el dominio
			Entendido	Entre 5 y 10 años en el dominio
			Novicio	Menos de 2 años en el dominio
	Capacidad de Articulación	Facilidad del educente para explicar su conocimiento	Alta	Explica sobresalientemente su conocimiento
			Media	Explica normalmente su conocimiento
			Baja	No se explica claramente su conocimiento
	Consenso Educentes	Grado de acuerdo inicial entre los educentes	Alto	Total consenso
			Bajo	No hay consenso
Dominio del Problema	Grado de Definición	Claridad de los objetivos del proyecto	Alto	Bien definido
			Bajo	Mal definido
	Nivel de Información Disponible	Tipo de información que se dispone previamente a la sesión, antes de aplicar la técnica	Táctico	Se cuenta con procedimientos, heurísticas
			Básico	Se cuenta con conceptos, atributos
			Nulo	No se cuenta con información
	Tipo Información a Educir	Tipo de información que la técnica puede educir	Estratégico	Se educe estrategias/control
			Táctico	Se educe procedimientos/heurísticas
			Básico	Se educe conceptos/atributos
Proceso de Educción	Restricción de Tiempo Proyecto	Disponibilidad suficiente de tiempo para aplicar la técnica	Alta	Mal de tiempo
			Media	Tiempo ajustado
			Baja	Holgura de tiempo
	Momento del Proceso	Etapa del proceso de educción en que se encuentra antes de la sesión	Inicial	Educción de definiciones generales
			Intermedio	Educción de requisitos principales
			Avanzado	Educción información final

Evaluación de Arquitecturas de Software con ATAM (Architecture Tradeoff Analysis Method): un caso de estudio

Andrea Delgado, Alberto Castro, Martín Germán
Universidad de la República, Facultad de Ingeniería, Instituto de Computación
{adelgado, acastro, mgerman}@fing.edu.uy

Resumen

La Arquitectura de Software condiciona las características del producto final en cuanto a cualidades como desempeño y mantenibilidad. El Architecture Tradeoff Analysis Method (ATAM) es una metodología para evaluar Arquitecturas de Software basada en los atributos de calidad especificados para el sistema, desarrollada por el Software Engineering Institute (SEI). El Grupo de Ingeniería de Software (Gris) del Instituto de Computación tiene como eje de sus actividades un programa de construcción y prueba de modelos de proceso, en cuyo contexto se desarrolló una herramienta para especificar modelos de procesos. ATAM se puso en práctica para evaluar el producto obtenido y realizar un segundo ciclo de desarrollo sobre el mismo, mejorando aspectos de riesgo identificados principalmente para el atributo de calidad performance. Se presenta ATAM y el caso de estudio realizado en el marco del programa.

1. Introducción

El eje de las actividades del grupo de Ingeniería de Software (Gris) del Instituto de Computación de la Facultad de Ingeniería de la Universidad de la República, Uruguay [1] está constituido por el programa de construcción y prueba de modelos de proceso. El programa inició en el año 2000 y utiliza como banco de pruebas de los procesos al curso "Proyecto de Ingeniería de Software" [2] que se dicta en cuarto año de la carrera de Ingeniería en Computación. En este curso los estudiantes desarrollan aplicaciones en proyectos sometidos a restricciones similares a las de la industria, contrastando la teoría con la práctica, donde clientes externos al curso plantean requerimientos para la obtención de aplicaciones de mediano porte con desafíos tecnológicos importantes. Si bien el objetivo central del programa es el desarrollo de modelos de proceso que puedan resultar adecuados para su transferencia a la industria, también se utiliza para desarrollar prototipos de aplicaciones en el marco de proyectos del Instituto de Computación y de la Facultad de Ingeniería, así como para empresas del medio. Una descripción completa del programa desde su concepción y evolución hasta el presente puede ser consultada en [3].

En el contexto del programa, en el año 2004 un docente del grupo de Ingeniería de Software (Gris) presentó un proyecto para la construcción de una herramienta que permitiera especificar modelos de procesos y generar en base a dicha especificación un documento con la descripción del proceso y un sitio web con la definición realizada, utilizando para las pruebas el proceso base del programa que es una adaptación del Rational Unified Process (RUP) [4].

Para el año 2005 se decidió realizar un segundo ciclo de desarrollo para evolucionar el prototipo de la herramienta obtenido, mejorando principalmente aspectos de performance, y agregando funcionalidades nuevas. En marzo del 2005 se propone por parte de otro docente del grupo de Ingeniería de Software (Gris), una pasantía para realizar la evaluación de la herramienta obtenida aplicando el Architecture Tradeoff Analysis Method (ATAM) [5] desarrollado por el Software Engineering Institute (SEI) [6], de forma de identificar los aspectos de riesgo que se deberían tener en cuenta en este segundo ciclo de desarrollo, y maximizar la reutilización de productos generados por dicho proyecto, además de evaluar la metodología de evaluación propuesta en ATAM como parte del estudio de la pasantía [7].

2. Architecture Tradeoff Analysis Method (ATAM)

El Architecture Tradeoff Analysis Method (ATAM) es una metodología para evaluar Arquitecturas de Software que principalmente evalúa la adecuación de la Arquitectura de Software definida con respecto a los atributos de calidad especificados para el sistema. Surge confluyendo ideas y técnicas de tres áreas: la noción de estilos o patrones de arquitectura, el análisis de atributos de calidad y el método Software Architecture Analysis Method (SAAM) [8] que es el predecesor del ATAM. Obtiene su nombre del concepto de que a veces no será posible cumplir con

todos los atributos de calidad definidos y se deberán realizar concesiones mutuas (o tradeoffs) entre estos, para obtener el balance adecuado. Existen también otros métodos para analizar arquitecturas como el Quality Attribute Worksop (QAW) [9], que surge para complementar el ATAM, y que identifica atributos de calidad importantes y clarifica los requerimientos del sistema antes de que exista una arquitectura definida que pueda ser evaluada con ATAM, y que si bien resulta más simple de aplicar, no es tan completo. Los métodos de evaluación de arquitecturas son bastante recientes, la Arquitectura de Software como tal surge en la década de los 90 y se populariza como área de investigación y práctica dentro de la Ingeniería de Software debido principalmente al crecimiento en la complejidad de los sistemas de software actuales incluyendo la variedad de nuevas tecnologías disponibles. Esto hace que cada vez sea más necesaria la definición explícita y documentación de la Arquitectura de Software, teniendo en cuenta la identificación de estilos o patrones de arquitecturas [10] que son utilizados comúnmente y presentan características específicas, y cuya aplicación trae aparejadas determinadas ventajas y desventajas en el contexto del problema que resuelven.

En el marco de ATAM se toma como definición de arquitectura la realizada en [11] que establece que la Arquitectura de Software de un programa o sistema de computación es la estructura o las estructuras del sistema, que contienen componentes de software, las propiedades externamente visibles de dichos componentes y las relaciones entre ellos. La Arquitectura de Software condiciona las características del producto final en cuanto a cualidades o atributos de calidad como performance y mantenibilidad, por lo que resulta importante poder evaluar el cumplimiento de los mismos en forma temprana para corregir errores antes de pasar a la codificación del sistema, donde es más costoso. Sin embargo, evaluaciones a posteriori resultan útiles como forma de aprendizaje y estudio de posibilidades de mejora, por ejemplo para desarrollar una nueva versión o para entender un sistema legado.

2.1. Conceptos de ATAM

En esta sección se describen los principales conceptos de la metodología ATAM según [5].

Para evaluar un diseño arquitectónico contra los requerimientos de atributos de calidad es necesario contar con una caracterización de los atributos de calidad. Por ejemplo para entender una arquitectura desde el punto de vista de la modificabilidad requiere entender como observar y medir este atributo, así como entender de que forma impactan en este atributo los distintos tipos de decisiones arquitectónicas. ATAM brinda una caracterización para varios atributos creada a partir del conocimiento existente en las comunidades de atributos de calidad, principalmente para atributos como performance, modificabilidad, disponibilidad, usabilidad y seguridad. Cada caracterización de atributos de calidad se divide en tres categorías: estímulo externo, decisiones arquitectónicas y respuestas, donde los primeros son eventos que hacen que la arquitectura responda o cambie, las decisiones arquitectónicas son los elementos de la arquitectura (componentes, conectores y sus propiedades) que tienen impacto directo en la obtención de respuestas a los atributos, y las respuestas se caracterizan por atributos medibles como latencia y throughput. Como se mencionó previamente, en general no se cuenta con una especificación precisa de los requerimientos de los atributos de calidad para los sistemas, por lo que en general la primer tarea a realizar en una evaluación de arquitectura es obtener una especificación precisa de los objetivos de calidad contra los cuales la arquitectura será evaluada. El mecanismo utilizado para obtener esta información es el Escenario.

Un escenario es una especificación corta que describe la interacción entre un stakeholder y el sistema, siendo la definición de stakeholder cualquier involucrado en el desarrollo, tanto cliente, como desarrolladores como gerentes, entre otros. Los escenarios proveen una forma de concretizar cualidades a tener en cuenta en tiempo de desarrollo como la modificabilidad, ayudan a entender cualidades de tiempo de ejecución como performance o disponibilidad, entre otros. Se definen tres tipos de escenarios: de Casos de Uso que involucran usos típicos del sistema y se utilizan para obtener información del mismo, similares a los definidos en la metodología de Casos de Uso que utiliza el RUP [4], escenarios de crecimiento que cubren el principio de anticipación al cambio teniendo en cuenta los posibles cambios futuros del sistema, y escenarios de exploración que cubren cambios extremos que se espera que estresen al sistema. Estos casos distintos de escenarios se utilizan para probar el sistema desde distintos puntos de vista, permitiendo que no se pasen por alto decisiones arquitectónicas que puedan representar riesgos.

La información de los escenarios es obtenida y priorizada en ATAM utilizando dos mecanismos diferentes en distintos momentos, y con la participación de diferentes stakeholders, estos son los árboles de utilidad y la lluvia de ideas estructurada. Los árboles de utilidad se realizan con el Arquitecto y proveen un mecanismo de enfoque top-down para traducir los principales requerimientos del negocio para el sistema, en escenarios concretos de atributos de calidad. Por ejemplo un requerimiento del tipo “la

seguridad es central para el éxito del sistema ya que los datos de los clientes son de extrema importancia" no es un objetivo específico de calidad que pueda ser evaluado, y debe ser traducido a una especificación más concreta y específica, además de entender la importancia relativa respecto de otros atributos de calidad como la performance, para determinar el foco de la evaluación. El árbol de utilidad se arma partiendo de un nodo raíz que representa la cualidad global en estudio, y a partir de ese nodo se van detallando los requerimientos de atributos de calidad hasta llegar a expresiones concretas en las hojas del árbol, que definen los escenarios para la evaluación. Estos escenarios se priorizan en dos dimensiones: por la importancia que tiene cada uno para el éxito del sistema, y por el grado de dificultad que posee para ser realizado, según estima el Arquitecto. En ambas dimensiones se utiliza habitualmente la escala High, Medium y Low. La salida de la generación del árbol de utilidad es una lista priorizada de los requerimientos de los atributos de calidad especificados como escenarios, esto permite que el equipo de evaluación se concentre en aquellas propuestas arquitectónicas involucradas en la satisfacción de los escenarios de alta prioridad que se encuentran calificados en las hojas.

La lluvia de ideas de escenarios se realiza con todos los stakeholders involucrados en el sistema, ya que el objetivo principal es encontrar todos los escenarios posibles según la clasificación vista anteriormente. Se ha comprobado que la lluvia de ideas en grupos numerosos permite crear una atmósfera donde las ideas de una persona estimulan al resto a pensar en esa dirección y otras, promueve la comunicación, la creatividad, y sirve para expresar el pensamiento colectivo de los participantes respecto al sistema. La lista obtenida por este medio se compara con el árbol de utilidad y se agregan a éste los nuevos escenarios.

A partir del árbol de utilidad y la priorización de escenarios obtenida, se evalúa la adecuación de las propuestas arquitectónicas presentes en la Arquitectura de Software en evaluación, respecto de los atributos de calidad especificados, identificando riesgos, no riesgos, puntos de sensibilidad y puntos de concesión. Los riesgos son decisiones arquitectónicas importantes que no se han tomado en el proyecto, o que si se han tomado pero cuyas consecuencias no son comprendidas totalmente. Los no riesgos son las fortalezas de la Arquitectura, aquellas decisiones de diseño arquitectónico tomadas que son adecuadas, es decir, cumplen con los requerimientos de calidad del sistema. Los puntos de sensibilidad son parámetros en la arquitectura con los cuales se relaciona fuertemente la respuesta a algún atributo de calidad medible, por ejemplo el rendimiento o throughput afectado por la velocidad de un canal de comunicación. Un punto de concesión (o tradeoff point) se encuentra en la arquitectura cuando un parámetro arquitectónico refiere a más de un punto de sensibilidad donde los atributos de calidad medibles son afectados en forma distinta según los cambios en dicho parámetro. Por ejemplo si se aumenta la velocidad del canal de comunicación se mejora el rendimiento o throughput pero se reduce la confiabilidad, entonces es un punto de concesión. Los tres aspectos mencionados, riesgos, puntos de sensibilidad y concesión, podrían ser el foco del esfuerzo futuro de prototipado, diseño y análisis.

2.2. Metodología de ATAM

El corazón de ATAM consiste en la ejecución de nueve pasos que se dividen en cuatro grupos que su vez, se realizan en el tiempo en cuatro Fases diferenciadas. Estos cuatro grupos no se corresponden uno a uno con las Fases, sino que se realizan en las Fases 1 y 2, y se agrega una Fase 0 previa de preparación y una Fase 3 posterior de finalización del proyecto. Si bien la numeración de pasos sugiere linearidad, la ejecución de los mismos no necesariamente es un proceso en cascada estricto, ya que se podrá volver a pasos anteriores o saltar hacia adelante a pasos posteriores, o inclusive iterar entre pasos según sea necesario. Los cuatro grupos en que se dividen los nueve pasos definidos consisten en un primer grupo de presentación, donde se intercambia información del sistema, un segundo grupo de investigación y análisis, donde se valoran los atributos de calidad claves uno a uno con las propuestas arquitectónicas, un tercer grupo de pruebas donde se revisan los resultados obtenidos contra las necesidades relevantes de los stakeholders, y un cuarto y último grupo donde se presentan los resultados del ATAM.

En la Fase 0 se acuerdan tiempo, fechas, costos, esfuerzo y se forma el equipo de evaluación, en las Fases 1 y 2 se realiza la evaluación con ATAM siguiendo los nueve pasos especificados, y finalmente en la Fase 3 se realiza el informe final de la evaluación realizada, se recoge información para la medida y mejora del proceso y se actualizan los repositorios de productos generados. A continuación se describen las Fases 1 y 2 que constituyen la base de ATAM.

2.2.1. Fase 1: En la Fase 1 se realizan las actividades correspondientes a los grupos de presentación e investigación y análisis. El grupo de presentación se compone de tres pasos: 1 – Presentar el ATAM, 2 – Presentar las pautas del negocio y 3 – Presentar la Arquitectura. En el primer paso el equipo de ATAM presenta el método a los stakeholders explicando el proceso a seguir y el involucramiento y responsabilidad de cada uno en el proyecto. Se detallan los pasos a seguir, las técnicas a utilizar y los

resultados a obtener. En el segundo paso un director de proyecto o gerente presenta el sistema desde el punto de vista del negocio, al equipo de ATAM y a los stakeholders, detallando principales funcionalidades, restricciones y metas definidas para el sistema. En el tercer paso el Arquitecto presenta la Arquitectura de Software definida incluyendo por lo menos los estilos utilizados, otros sistemas con que se debe interactuar, restricciones técnicas de software como por ejemplo uso de sistema operativo. Si se cuenta con un documento de Software Architecture Description (SAD) el Arquitecto debería basar su explicación en las distintas vistas contenidas en el mismo.

El segundo grupo de investigación y análisis se compone también de tres pasos: 4 – Identificar las propuestas arquitectónicas, 5 – Generar el árbol de utilidad de los atributos de calidad y 6 – Analizar las propuestas arquitectónicas. En el cuarto paso el equipo de ATAM le pide al Arquitecto que identifique las propuestas arquitectónicas o estilos de arquitectura utilizados, ya que éstos definirán las estructuras importantes definidas para el sistema y las características implicadas. En el quinto paso se genera el árbol de utilidad donde principalmente el Arquitecto pero también los principales stakeholders, identifican, priorizan y refinan los requerimientos de atributos de calidad más importantes del sistema, según se mencionó previamente, identificando los escenarios en el árbol y su importancia en las dos dimensiones definidas. En el sexto paso se analizan las propuestas arquitectónicas según el árbol de utilidad generado, esto es, que tan adecuados son el uno para el otro, evaluando como cada propuesta arquitectónica influye en la obtención o no del atributo de calidad requerido, e identificando los riesgos, no riesgos, puntos de sensibilidad y concesión asociados a dicha evaluación.

2.2.1. Fase 2: En la Fase 2 se realizan las actividades incluidas en el tercer grupo de pruebas y el cuarto grupo de informes, que completan los tres pasos restantes. El grupo de pruebas se compone de dos pasos: 7 – Lluvia de ideas y 8 – Analizar las propuestas arquitectónicas y el grupo de Informes se compone de un solo paso: 9 – Presentar los resultados. En el séptimo paso se confirman e identifican nuevos escenarios según varios stakeholders involucrados, los que también se priorizan y se comparan con los identificados en el árbol de utilidad. Pueden suceder tres casos: el escenario ya está en el árbol de utilidad, no está pero encaja en alguna rama y se convierte en una nueva hoja, no está y no encaja en ninguna rama, lo que significa que no había sido considerado previamente. En el octavo paso se realiza lo mismo que en el sexto paso para el nuevo árbol de utilidad con todos los escenarios incluidos. Finalmente en el noveno paso el equipo de ATAM presenta los resultados a los stakeholders y entrega la documentación correspondiente a las salidas del ATAM: documento de propuestas arquitectónicas, conjunto de escenarios priorizados, conjunto de preguntas basadas en los atributos, árbol de utilidad, los riesgos descubiertos, los no riesgos documentados, los puntos de sensibilidad y de concesión encontrados, más la relación entre los riesgos encontrados y su impacto en las pautas del negocio definidas.

3. Aplicación de ATAM

La aplicación de la metodología de ATAM para evaluación de Arquitecturas de Software se realizó en el contexto de las actividades del Grupo de Ingeniería de Software (Gris) en el marco del programa de construcción y prueba de modelos de proceso de aplicación en el curso "Proyecto de Ingeniería de Software". La pasantía realizada por dos estudiantes de quinto año de la carrera comenzó a mediados del año 2005 y tenía como objetivos el estudio y aplicación de ATAM para evaluar la herramienta de definición y generación de modelos de proceso de desarrollo de software construida por un grupo de estudiantes en el curso del año 2004.

En dicho curso un docente del Gris cumplió el rol de cliente especificando los requerimientos para el desarrollo de la herramienta, que debía permitir como mínimo la definición y generación del proceso base adaptación del RUP que se utiliza en el curso. Esta herramienta debía permitir la especificación de distintos elementos de modelado como ser fases, hitos, disciplinas, actividades, roles y entregables definidos para el modelo de proceso. Asimismo resultaba interesante la posibilidad de permitir que los entregables generados en el marco de la realización del proceso por los estudiantes, pudieran ser enviados a través de una página web y gestionados en la instanciación del proceso que se estuviera probando en esa edición del curso. El producto obtenido denominado Graphead [12] tenía como fortaleza la generación del sitio web que resultaba muy completa y amigable, pero presentaba problemas de performance en la especificación del modelo de proceso al trabajar con los elementos del mismo, y no incluía manejo de las distintas versiones que se especifican para cada año.

Dado el interés que tiene para el Grupo de Ingeniería de Software (Gris) contar con dicha herramienta para la definición de distintos modelos de proceso para cada curso, se decidió realizar un segundo ciclo de desarrollo para mejorar aspectos del prototipo obtenido y agregar nuevas funcionalidades. La pasantía de aplicación de ATAM se definió como forma de apoyar este segundo ciclo de desarrollo para que se

contara también como entrada del mismo, con el informe de resultados de la evaluación de la Arquitectura de Software del primer producto.

3.1. Caso de estudio con ATAM

Para la realización de la pasantía se definieron cinco etapas que planteaban: el estudio de ATAM, el estudio de la herramienta a evaluar y su documentación, la aplicación de ATAM para evaluar la Arquitectura de Software de la herramienta, la presentación de resultados de la evaluación a los docentes del Grupo de Ingeniería de Software (Gris) y al grupo que realizaría el segundo ciclo de desarrollo, y por último el cierre de la pasantía con la entrega de la documentación final de cada etapa, cuyas primeras versiones debían entregarse al finalizar cada etapa como parte del hito definido para la misma.

La evaluación elegida fue realizar un ciclo del ATAM en su formato tardío. Esta versión del método permite entender al producto en su conjunto, y dado que se planeaba realizar un segundo ciclo de desarrollo del mismo, una evaluación tardía es especialmente indicada para entender sistemas legados. Una vez elegido el método, se tuvieron que tomar decisiones sobre como se realizaría el desarrollo del mismo. La realidad estaba dada por la ausencia del equipo de desarrollo del producto en cuestión, solo contando para el entendimiento del sistema con su documentación. El único stakeholder con el que se contaba era el cliente con el cual se podía interactuar en los pasos que así lo requirieran. Este escenario marcó el desarrollo de la evaluación, adaptando los pasos del ATAM a los insumos existentes.

En la Fase 1 se pudieron realizar los pasos 1 y 2 del grupo de presentación, y el paso 3 al no contar con el Arquitecto del producto se realizó estudiando la documentación del sistema, en particular el Software Architecture Document (SAD) y el Modelo de Diseño. Del grupo de investigación y análisis se realizaron los pasos 4, 5 y 6, este último con bastante dificultad dada la falta del Arquitecto para justificar las elecciones de diseño realizadas. De la Fase 2 el grupo de pruebas no se realizó ante la falta de stakeholders del proyecto, si se hizo el grupo de informes que fue presentado a los interesados como estaba previsto.

3.1.1. Fase 1 – grupo de presentación: se realizaron los pasos correspondientes a este grupo, comenzando por el paso 1 de presentación del ATAM que se hizo para los docentes del Grupo de Ingeniería de Software (Gris) entre los cuales se encontraba el cliente del producto en evaluación. Una de las instancias en la que la participación del cliente es clave es el paso 2 de presentación de las pautas del negocio. Durante la misma, el cliente explicó la finalidad que tiene el producto, y los aspectos de Graphead que no le satisficieron. Del mismo modo, transmitió cuales eran sus expectativas de la evaluación, y también como esperaba que esta ayudara a mejorar la calidad del producto en la próxima iteración de mismo. Fue en esta reunión donde el equipo de evaluación obtuvo los requerimientos de los atributos de calidad que el tenía del sistema, los cuales guiarían el resto de la evaluación. Además se estudió la documentación del producto Graphead de la disciplina Requerimientos: el Modelo de Casos de Uso, Especificación de Requerimientos, Glosario y Modelo de Dominio. Una vez conocidas las pautas del negocio, el equipo de evaluación debió enfrentarse a la Arquitectura de Software del sistema por si solo, dado que como ya fue mencionado no se contaba con la presencia del arquitecto. Por lo tanto no se realizó el paso de presentación de la Arquitectura.

3.1.2. Fase 1 – grupo de investigación y análisis: para poder identificar y entender las propuestas arquitectónicas realizadas por el Arquitecto según lo que se indica en el paso 4 de ATAM, se recurrió a la documentación existente. Se estudiaron varios documentos, principalmente el Software Architecture Document (SAD) y el Modelo de Diseño, donde en el primero se presenta la Arquitectura definida en base a los Casos de Uso identificados como relevantes. A partir de este estudio se obtuvo una visión general de la Arquitectura y de las principales decisiones tomadas, por ejemplo en lo que tiene que ver con los estilos o patrones utilizados y la descomposición en subsistemas realizada. En la figura 1 se muestra el diagrama general de la Arquitectura definida, que sigue un estilo en capas relajado, esto es, capas superiores acceden a capas inferiores no inmediatas:

Figura 1. Estilo en capas utilizado en la Arquitectura de Graphead

La Interfaz de Usuario tiene como objetivo el manejo de la lógica del usuario y contiene el conjunto de componentes de Jgraph [13] y módulos que encapsulan la lógica de los casos de uso. Los Servicios del Sistema representan los servicios básicos que debe proveer el sistema los que son directamente utilizados por los módulos de la capa superior, constituyendo la entrada a la lógica del sistema. Los subsistemas que contiene ofrecen las interfaces requeridas por cada caso de uso asociado al subsistema y los controladores que las implementan. Los Servicios de Negocio son servicios de manejo de información del negocio, aun más básicos que los de la capa superior, lo que permite reutilizar estos módulos en otros subsistemas. En la

capa de infraestructura se ubican módulos adaptadores y de servicio general, útiles para cualquiera de los subsistemas, como es el encapsulador del DBMS utilizado. Luego se describen los subsistemas definidos en cada capa y las responsabilidades asociadas a cada uno, así como los COTS utilizados en el desarrollo.

Sin embargo, la documentación de la Arquitectura debería cubrir, según ATAM, las restricciones técnicas, otros sistemas con los cuales se debe interactuar y las propuestas arquitectónicas utilizadas para alcanzar los requerimientos de los atributos de calidad. ATAM centraliza el análisis de una arquitectura en el entendimiento de sus propuestas arquitectónicas. Se concentra en identificar propuestas y estilos arquitectónicos porque estos representan la manera en que la arquitectura cumple con los atributos de calidad, esto significa que se asegura que los requerimientos críticos serán alcanzados en forma predecible. Estas propuestas arquitectónicas definen las estructuras importantes del sistema, y describen como el sistema puede crecer, responder a cambios, entre otros.

Luego de analizar la documentación existente del sistema, siguiendo las premisas propuestas por ATAM, se descubrieron distintas falencias en dicha documentación que influyen negativamente en el posterior desarrollo de la evaluación, entre las que se pueden destacar: 1) los casos de uso relevantes para la arquitectura en el SAD no se sabe porque fueron seleccionados, no existe una justificación para la elección de los mismos. 2) no se documentan los componentes que encapsulan las comunicaciones con software externo, en particular el uso del framework Hibernate [14] es oscuro. No se sabe como se maneja la actualización de datos a través del mismo, cuando según el Modelo de Diseño todas las colaboraciones entre componentes allí existentes, realizan persistencia. 3) no se específica que componentes encapsulan el manejo del resto del software externo utilizado, según la presentación del producto, como ser Castor[15] para mapeo de clases java y archivos XML, Xalan[16] para generación de HTML a partir de XML mediante transformación XSL[17], Fop[18] para generación de PDF a partir de XML mediante transformación XSL, y Jasper[19] para generación de la agenda del proceso en formatos html, pdf y xls. 4) el Modelo de Diseño, así como el SAD, no refina los componentes críticos del sistema con la especificidad necesaria como para entender como se satisfacen los requerimientos. En particular, no se sabe como se alcanzan los requerimientos de los atributos de calidad. 5) Los requerimientos de calidad deseados por el cliente, como ser la performance, no se encuentran especificados. 6) La Documentación Técnica, es solo un listado de paquetes e interfaces. No se conoce el comportamiento de cada clase o interfaz.

En vista de la presente dificultad para comprender en detalle las propuestas arquitectónicas descritas en la documentación, el equipo de evaluación gestionó una entrevista con el Arquitecto, la que finalmente pudo ser realizada. En la misma se obtuvo valiosa información que no se desprendía en forma directa de la documentación evaluada. Consideramos que tuvo una influencia positiva en el resultado de la evaluación, y confirma porque para ATAM es tan importante la participación de los distintos stakeholders, en particular el Arquitecto, en la evaluación.

El paso 5 de generación del árbol de utilidad se realizó en base a los requerimientos de los atributos de calidad obtenidos en la entrevista con el cliente. Este árbol contiene los escenarios que ponen a prueba a las propuestas arquitectónicas, para ver si estas alcanzan los requerimientos de los atributos de calidad especificados. Para realizar este estudio se generaron veintidós escenarios distintos que modelan los requerimientos de los atributos de calidad especificados por el cliente. Cada uno de estos escenarios fue ubicado en una hoja del árbol de utilidad, según el atributo de calidad que estuviera poniendo a prueba. Teniendo en cuenta la importancia de cada escenario para el éxito del sistema dada por el cliente se asignó la priorización en la primera dimensión, y según el grado de dificultad estimado por el grupo de evaluación para realizarlo, se asignó la priorización en la segunda dimensión. En la figura 2 se muestra una selección de los escenarios generados, para algunos de los atributos de calidad especificados:

Atributo de Calidad	Refinamiento del Atributo	Escenarios
Performance	Tiempo de respuesta	Informar de la aceptación de la carga de un proceso dado en menos de 0.3 segundos **(M,L)**
	Throughput	Versionar un proceso con 300 elementos en menos de 10 segundos **(H,H)**
	Latencia	Cargar un proceso con 300 elementos en menos 3 minutos **(H,M)**
	Carga de trabajo	Agregar un nuevo elemento a un proceso de 300 elementos, no aumentará los tiempos de latencia en más de un 15%**(H,H**
Maintainability	Cambios en un subsistema no requiere cambios en otros subsistemas	Cambiar el motor de la base de datos implica cambiar un subsistema **(L,M)**
		Cambiar un COTS no requiere cambiar más de un subsistema **(L,M)**
Modifiability	Cambiar COTS	Cambiar el motor de la base de datos en menos de 10 día/persona **(L,M)**
		Cambiar el generador web en menos de 5 día/persona **(L,M)**
		Cambiar el generador pdf en menos de 5 día/persona **(L,M)**
Reusability	Reusabilidad de componentes	Los conceptos del negocio son utilizables por distintas aplicaciones **(M,M)**

Figura 2. Árbol de utilidad generado con ATAM

Los escenarios generados fueron ponderados y posteriormente priorizados por el equipo de evaluación, según los lineamientos marcados por el cliente. Una vez ordenados por la prioridad elegida, del total de los mismos se seleccionaron seis que fueron

validados con el cliente, para confirmar que el esfuerzo de la evaluación estaba bien enfocado según sus expectativas. Una vez obtenidos los escenarios priorizados se pudo comenzar con la siguiente etapa de análisis de propuestas arquitectónicas, para evaluar el cumplimiento o no de los atributos de calidad especificados en los escenarios priorizados. En la figura 3 se muestran los seis escenarios priorizados:

Ranking	Escenarios	Observaciones
1	Versionar un proceso con 300 elementos en menos de 10 segundos	Performance
3	Agregar un nuevo elemento a un proceso de 300 elementos, no aumentará los tiempos de latencia en más de un 15%	Performance
4	Cargar un proceso con 300 elementos en menos 3 minutos	Performance
5	Informar de la aceptación de un alta de actividad en menos de 0.3 segundos	Performance
6	Los conceptos del negocio son utilizables por distintas aplicaciones	Reusability

Figura 3. Escenarios priorizados para la evaluación con ATAM

En el paso 6 de análisis de propuestas arquitectónicas se evaluaron diferentes propuestas arquitectónicas para los escenarios priorizados. Estas propuestas intentan satisfacer los escenarios que describen los requerimientos de los atributos de calidad. No todas las propuestas analizadas fueron realizadas por el arquitecto, sino que la mayoría fueron elaboradas por el equipo de evaluación debido a las dificultades mencionadas sobre la documentación y a pesar de la entrevista con el Arquitecto. En el análisis de cada propuesta se buscó determinar sobre que atributos de calidad influía y que riesgos o no riesgos introducía, así como los puntos de sensibilidad y de concesión identificados en cada caso.

Por ejemplo para el escenario 1 – Versionar un proceso de 300 elementos en menos de 10 segundos, que corresponde al atributo de calidad general performance en su refinamiento throughput, se identificaron dos propuestas arquitectónicas relativas a la forma de acceder a los datos para realizar el versionado, cuyo análisis se resume en la figura 4:

Análisis de una Propuesta Arquitectónica					
Escenario #: 1	**Escenario**	Versionar un proceso con 300 elementos en menos de 10 segundos			
Atributo	Performance – Throughput				
Entorno	Proceso con 300 elementos				
Estímulo	Se invoca el versionado				
Respuesta	Versiona el proceso en menos de 10 segundos				
Decisión Arquitectónica		**Sensitivity**	**Tradeoff**	**Riesgo**	**No riesgo**
Cada subsistema accede directamente a la BD		S1	T1,T2	R1,R2	
Un subsistema que encapsule el acceso a datos (p.e.: uso de Hibernate)		S2	T3		
Razonamiento	S1. Disminuye la confiabilidad. S2. Aumenta la confiabilidad. R1. Aumenta el riesgo que los datos queden inconsistentes ante una eventual falla. R2. Aumento de dependencia entre la aplicación y la BD. T1. Mejora la performance, pero disminuye la maintainability. T2. Mejora la performance, pero disminuye la reusability. T3. Aumenta la reusability, pero empeora la performance.				

Figura 4. Análisis de propuestas arquitectónicas del escenario 1

En el primer caso se accede directamente desde cada subsistema y en el segundo caso se encapsula el acceso en un subsistema al que accede el resto. Para el primer caso se identificaron dos riesgos, uno la posibilidad de que los datos quedaran inconsistentes ante una eventual falla, y otro que se aumentaba la dependencia del sistema a la base de datos elegida. Como puntos de concesión del atributo performance evaluado se vio que empleando esta opción se mejoraba la performance pero disminuían la mantenibilidad y la reusabilidad, ya que por ejemplo, cambiar de manejador de base de datos requeriría un esfuerzo importante. En el segundo caso no se identificaron riesgos, como punto de concesión se evaluó que se incrementaba la reusabilidad y mantenibilidad, pero se disminuía la performance. Se identificó como punto de sensibilidad para las dos propuestas evaluadas, la confiabilidad, que en la primer opción disminuye y en la segunda aumenta.

Además de la evaluación de las propuestas arquitectónicas frente a los escenarios priorizados, se obtuvieron resultados adicionales debido al estudio de la documentación y de la entrevista con el Arquitecto en la cual se conocieron detalles de la implementación, donde según él radicaban la mayoría de los problemas asociados a la degradación de performance detectada. Estos resultados fueron también de utilidad para apoyar al grupo realizando el segundo ciclo de desarrollo, destacando los relacionados a problemas en el acceso a datos mediante Hibernate por un uso no del todo adecuado del framework, la decisión de actualizar la base de datos en tiempo real en lugar de aprovechar el caché mientras se está trabajando con un proceso y persistir al final del mismo, y la utilización de algunos algoritmos que no alcanzaban tiempos de ejecución adecuados. Por otro lado, se rescataba como de gran utilidad para el desarrollo el uso de otros componentes COTS como ser Castor, Xalan, Fop, Jasper y Jgraph.

3.1.3. Fase 2 – grupo de pruebas: debido a la realidad a la cual se enfrentaba el equipo de evaluación, se omitió la instancia en la cual todos los stakeholders proponen escenarios, y luego los analizan junto con los evaluadores. ATAM considera que a pesar de que los evaluadores a esta altura llevan un buen tiempo trabajando sobre la arquitectura y la conocen en detalle, siempre quedan algunas características ocultas, las cuales pueden llegar a ser críticas para alcanzar los requerimientos de los atributos de calidad. Es por este motivo que se introduce este paso. Por lo tanto, al no realizarlo, se perdió la oportunidad de descubrir nuevos puntos de sensibilidad, de concesión, riesgos y no riesgos que pudieran haberse pasado por alto.

3.1.4. Fase 2 – grupo de informes: para este paso se debió realizar nuevamente una adaptación sobre lo que se indica en ATAM, ya que se debían presentar los resultados a dos grupos distintos, por un lado el Grupo de Ingeniería de Software (Gris) interesado tanto en la

evaluación del producto Graphead como en la metodología ATAM y las adaptaciones realizadas a la misma en el marco de la pasantía. Por otro lado, al grupo de estudiantes que iba a realizar el segundo ciclo de desarrollo sobre el producto, a quienes interesaban principalmente las guías que se pudieran dar sobre la Arquitectura de Software evaluada y los aspectos a tener en cuenta según los riesgos, no riesgos, puntos de concesión y de sensibilidad encontrados, así como las posibilidades de reuso existentes, no siendo factible entrar en detalles de la metodología ATAM aplicada. Por lo tanto, se hicieron dos presentaciones, una como indica ATAM entrando en los detalles de la evaluación realizada, y otra orientada a la conceptualización de los resultados obtenidos en la evaluación. Las principales observaciones fueron las correspondientes a los problemas detectados en el manejo de la persistencia, recomendando realizar nuevamente este desarrollo, y el de adecuación del resto de los COTS utilizados, en particular el generador del sitio web y documento pdf del proceso, recomendando la reutilización de dicho componente y los COTS que éste incluía.

5. Conclusiones y trabajo futuro

Como resultado de la aplicación del método se obtuvo una lista de escenarios priorizados y las diferentes propuestas arquitectónicas que los intentan satisfacer, a partir de las cuales se descubrieron puntos de sensibilidad, de concesión, y riesgos asociados. Estas propuestas arquitectónicas son una excelente guía para tomar futuras decisiones arquitectónicas, ya que se analizó su impacto en la Arquitectura, y atacan directamente las expectativas de calidad que el cliente tiene del sistema.

Consideramos que el método resultó muy útil para evaluar la Arquitectura de Software, a pesar de que no se siguió la metodología propuesta en forma estricta, incluso aún omitiendo y modificando pasos según las posibilidades con que se contaba. La evaluación permitió un enfoque amplio del sistema, en lugar de uno estrecho o con miras a corto plazo. Gracias a este tipo de enfoque que plantea el método de evaluación se descubrieron riesgos ocultos y errores que explican los problemas que tiene el producto.

Creemos que ATAM se puede adaptar a las necesidades particulares de cada negocio, como se hizo en este caso. El segundo ciclo de desarrollo fue exitoso y se pudieron mejorar los problemas de performance del producto, manteniendo los aspectos positivos detectados reutilizando los componentes mencionados. Una descripción de este producto puede verse en [20].

Como trabajo a futuro se piensa incluir una adaptación de método ATAM en las actividades de la Disciplina de Aseguramiento de la Calidad del proceso base adaptación del RUP del curso “Proyecto de Ingeniería de Software”, que permitan realizar una evaluación de la Arquitectura de Software definida antes de que sea implementada. Creemos que realizar esta evaluación permitirá encontrar puntos de sensibilidad, de concesión, de riesgos y no riesgos para las propuestas identificadas que aportarán a la mitigación de riesgos en las decisiones de diseño tomadas, antes de la implementación de la Linea Base de la Arquitectura en la Fase de Elaboración.

Luego del estudio realizado por la pasantía, el método ATAM se incluyó en el temario de la asignatura electiva de quinto año de la carrera “Taller de Arquitectura de Software”[21] dictada por el Grupo de Ingeniería de Software (Gris) en el marco de las áreas de investigación del grupo.

6. Referencias

[1] Grupo de Ingeniería. de Software (Gris), Instituto de computación, Facultad de Ingeniería, Universidad de la República http://www.fing.edu.uy/inco/grupos/gris/

[2]Proyecto Ingeniería de Software,Instituto de Computación Facultad de Ingeniería, Universidad de la República http://www.fing.edu.uy/inco/cursos/ingsoft/pis/index.htm

[3] Pérez, B. & Delgado, A., Modelo de Desarrollo de Software OO – Experimentación en un curso de Ingeniería de Software, JIISIC 2006, Puebla, México.

[4] IBM Rational Unified Process. http://www-130.ibm.com/developerworks/rational/products/rup/

[5] Clements P., Kazman R., Klein R., Evaluating Software Architectures, Methods and Case Studies, Addison-Wesley Pearson Education, 2004

[6] Software Engineering Institute http://www.sei.cmu.edu/

[7] Castro A., Germán M., Pasantía de ATAM http://www.fing.edu.uy/inco/cursos/ingsoft/atam.htm

[8] SAAM http://www.sei.cmu.edu/publications/articles/saam-metho-propert-sas.html

[9] QWA http://www.sei.cmu.edu/architecture/qaw.html

[10] Shaw M., Garlan D., Software Architecture: perspectives on an emerging discipline, Prentice-Hall, 1996.

[11] Bass L., Clements P., Kazman R., Software Architecture in practice, Addison-Wesley, 2004

[12] Proyecto de Ingeniería. de Software, Memoria Organizacional, Universidad de la República, Facultad de Ingeniería, Instituto de Computación, Grupo 3 año 2004 http://www.fing.edu.uy/inco/cursos/ingsoft/pis/index.htm

[13] Jgraph http://sourceforge.net/projects/jgraph/

[14] Hibernate http://www.hibernate.org/

[15] Castor www.**castor**.org/

[16] Xalan http://xalan.apache.org/

[17] XSL http://www.w3.org/TR/xsl/

[18] Fop http://xmlgraphics.apache.org/fop/

[19] Jasper http://jasperforge.org/sf/projects/jasperreports

[20] Pérez, B., DeVeloPro, Herramienta para la Documentación y Gestión de los Procesos de una Organización, CACIC 2006, San Luis, Argentina.

[21]Taller de Arquitectura de Software, Instituto de computación, Facultad de Ingeniería, Universidad de la República http://www.fing.edu.uy/inco/cursos/tarqsoft

Agradecimientos

El presente trabajo ha sido desarrollado en el marco del proyecto COMPETISOFT (Mejora de Procesos para Fomentar la Competitividad de la Pequeña y Mediana Industria de Software de Iberoamérica) del programa CYTED (Ciencia y Tecnología para el Desarrollo).

Transformación de Vistas Arquitectónicas Orientada por Modelos

Rogelio Limón Cordero[1], Isidro Ramos Salavert[1], Arturo Aragon Sorroza[2]
[1]*Departamento de Sistemas y Computación, Universidad Politécnica de Valencia,*
Camino de Vera s/n E-46071 Valencia – España
{rlimon,iramos}@dsic.upv.es
[2]*Centro de Computo, Instituto Tecnológico de Oaxaca*
Av. Victor B. Ahuja 125 esq. Calz. Tecnológico, Oaxaca, México
arturoar@itoaxaca.edu.mx

Resumen

Las vistas arquitectónicas permiten la representación y documentación de las estructuras estáticas y dinámicas de la arquitectura de software de sistemas complejos, son "los planos arquitectónicos" del sistema con los cuales se logran hacer análisis previos al diseño detallado, y posterior a la implementación realizar seguimientos en la evolución y mantenimiento del sistema. Sin embargo dada la diversidad de propuestas de modelos existentes para las vistas arquitectónicas se requiere establecer vínculos entre éstas para lograr, por un lado emigrar entre un modelo y otro, y por otro lado generar varias instancias arquitectónicas de una manera más rápida. Por tal motivo se propone llevar a cabo la transformación de los modelos que soportan las vistas arquitectónicas utilizando una transformación entre éstas mediante la Arquitectura Orientada por Modelo (MDA: Model Driven Architecture) propuesta por el OMG, que posibilita la transformación entre modelos que están al mismo nivel o a diferente nivel de abstracción.

1. Introducción.

Las vistas arquitectónicas permiten separar los diferentes asuntos de interés que se presentan al desarrollar un sistema de software, modelando las estructuras de la arquitectura. Lo cual permite tomar decisiones oportunas antes del diseño detallado y constituye una de las primeras referencias cuando se realizan tareas de mantenimiento o cuando el sistema evoluciona.

Cada estructura del sistema que es representada por una vista arquitectónica establece correspondencias entre sí que las vincula por ser parte del mismo sistema. Sin embargo estas correspondencias no son claramente establecidas por los distintos enfoques que modelan a las vistas, y mucho menos se conservan a estos vínculos en la especificación arquitectónica.

Por lo que la propuesta que se presenta en este trabajo va encaminada a la especificación de las correspondencias a nivel de meta-modelos arquitectónicos de una vista con otra, con el propósito de mantener una traza entre los meta-modelos de las vistas mediante la cual se pueda generar el modelo de una vista a partir de un modelo de otra vista. Esto se propone hacerlo mediante la arquitectura orientada por modelos (MDA).

El desarrollo orientado por modelos se está consolidando como un nuevo paradigma de desarrollo a través del MDA, que ofrece posibilidades de desarrollo independiente de la plataforma tecnológica y re-uso, realizando transformación entre modelos que pueden estar a un mismo nivel o a diferente nivel de abstracción. El MDA proporciona el soporte y los mecanismos para la transformación a nivel de meta-modelos entre enfoques de modelado de software distinto.

El trabajo que se presenta está organizado de la siguiente manera, en la sección 2 se comienza por clarificar y analizar la relación entre el MDA con la arquitectura de software (AS), puesto que las dos áreas tratan con modelos (arquitectónicos) empleando incluso términos semejantes (arquitectura, vistas, modelos). En la sección tres se hace un esbozo de la propuesta. En la sección cuatro se exponen los diferentes modelos de vistas arquitectónicas que se han realizado, las relaciones que hay entre ellos y se

presentan los meta-modelos de dos vistas usando la notación de UML 2.0. En la sección cinco se expone el soporte para llevar a cabo las transformaciones mediante un lenguaje (QVT) y en una notación gráfica se especificarán las correspondencias entre los meta-modelos de las dos vistas modeladas (modular y componente-conector). En la sección seis se ilustra con un caso de estudio y por último se exponen las conclusiones.

2. El desarrollo orientado por modelos y la arquitectura de software.

El desarrollo de software mediante la ingeniería orientada por modelos (MDE: Model-Driven Engineering) es una propuesta para enfrentar a la complejidad introducida por la diversidad tecnológica existente, cuya estrategia consiste en capturar los elementos de un modelo mediante meta-modelos, y genera otros al mismo o diferente nivel de abstracción mediante un mecanismo de transformación entre éstos [1].

Una de las propuestas de MDE hecha por el OMG (Object Management Group) es el MDA [2] la cual tiene como soporte a MOF para la especificación de meta-modelos y llevar a cabo las transformaciones entre modelos. Este enfoque al igual que la AS modela arquitecturas pero con diferente intencionalidad, aunque existan algunas semejanzas entre ellas.

2.1. MDA y AS.

La diferencia entre el MDA y la AS parte desde el concepto que maneja sobre arquitectura y su diferencia se hace más evidente en los propósitos que persigue cada área.

En cuanto al concepto de arquitectura, el MDA hace una generalización de la referencia que toma de [3] como base. Considera como arquitectura a cualquier modelo que especifique las "partes" del sistema y las reglas que establecen sus interacciones entre ellas mediante conectores [2], lo cual debe ser expresado formalmente o a través de una semántica claramente definida como por ejemplo el UML 2.0.

En el caso de la AS, la arquitectura es considerada como *el conjunto de estructuras que forman parte del sistema, cuyos elementos de software ("partes") sólo muestran sus propiedades externamente visibles y estos elementos están relacionados entre sí* [4].
De tal manera que si se ocupa por ejemplo un modelo como el *entidad-relación* (ER), éste es considerado como una arquitectura por el MDA donde las "partes" son las *entidades* y *atributos*, sus conectores e interacciones son sus ligas de *relaciones,* y las reglas son las que definen cómo deben establecerse las *relaciones*, en cambio en la AS este modelo ER no es considerado como una AS, puesto que sus elementos son de un nivel de abstracción superior al del software. Sin embargo un modelo de clases (orientación a objetos) es considerado como arquitectura en ambos enfoques.

Es importante señalar que cualquier arquitectura de software es considerada como una arquitectura en el MDA, pero no todas las arquitecturas del MDA son consideradas como arquitecturas de software.

En cuanto al propósito que persigue, el MDA tiene como principal objetivo el desarrollo del software usando a los modelos como elementos clave para el desarrollo y a la transformación entre modelos como el mecanismo para llevar a cabo el desarrollo.

En cambio la AS es ocupada para crear modelos del sistema (arquitecturas) con el fin de tomar decisiones tempranas antes de realizar el diseño del software, y servir como una guía para el desarrollo, mantenimiento y evolución del sistema.

Por lo tanto estas dos áreas son complementarias y pueden establecer una simbiosis entre ellas. La dos están siendo abordadas intensivamente en la actualidad como lo demuestran las publicaciones sobre el MDA en [5] y sobre la AS en [6].

En este trabajo se muestra cómo aplicar la estrategia de modelado del MDA para generar transformaciones entre vistas arquitectónicas, y cómo también es posible llegar a producir arquitecturas a niveles de abstracción cercanas a las del código.

2.2. Las transformaciones y la arquitectura de software.

Uno de los elementos claves del MDA es la transformación de modelos, con lo cual se logra pasar de un modelo de mayor nivel de abstracción a uno de menor nivel de abstracción, es decir modelar arquitecturas que son independientes de las plataformas llamadas modelos PIM (Platform Indepent Model) y llegar a generar modelos que se adecuen a la plataforma donde se ejecutará la aplicación, llamados modelos PSM (Platform Specific Model), tal como se muestra en a figura 1(a).

En la figura citada se observa también cómo se incorpora la información de una plataforma específica o información adicional (IA). De esta manera a partir de un PIM se pueden generar varios PSM. La tarea de modelado se concentra ahora sólo en el espacio del problema. Cuando se deseen obtener versiones para *x* plataforma se deberá aplicar una transformación incorporando la información de esa plataforma *x*. Es posible también no incorporar ninguna información de ninguna plataforma ni otra adicional, en todo caso la

transformación generará otro modelo PIM, es decir un modelo del mismo nivel de abstracción.

De la misma manera que se logra una transformación de un PIM a un PSM, es posible transformar un modelo de la AS partiendo de un modelo para un sistema que no depende de los detalles específicos de la aplicación (MAS) a un modelo arquitectónico de software generado para una aplicación específica (MASA), como se observa en la figura 1(b), que al realizar la transformación se incorporan los detalles de la aplicación. Vale la pena aclarar que siempre que se diseña una AS ésta es del tipo MAS (antes de diseño del sistema), y el modelo MASA corresponde a una AS de un sistema a nivel de diseño (a un nivel inferior de abstracción). También es posible generar modelos que están a un mismo nivel de abstracción, al no incorporar los detalles de la aplicación la transformación produciría otro MAS, esto es útil cuando se realizan transformaciones de vistas arquitectónicas del mismo nivel.

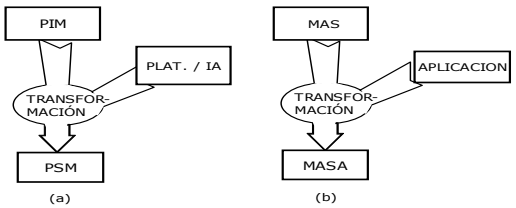


Fig. 1 Transformación de modelos (a) MDA (b) AS

3. La propuesta de transformación entre vistas arquitectónicas de software.

3.1 Los elementos de la transformación.

Para poder crear y realizar transformaciones de modelos MDA cuenta con el soporte de MOF 2.0 y de Query/Views/Transformations [2]. MOF proporciona un marco para la especificación del modelado y la notación para la representación de los modelos, y QVT es el lenguaje para realizar relaciones y hacer transformaciones.

MOF propone 3 niveles de abstracción por arriba de las instancias a modelar: el nivel 0 es el inferior y representa los objetos (instancias) a modelar. Por ejemplo una factura: el nivel 1 es el uso de un modelo para representar a los elementos físicos, como las tablas del modelo relacional que represente los datos de x facturas; el nivel 2 es la especificación del modelo usado en el nivel 1, es decir un meta-modelo; y el nivel 3 es el lenguaje (MOF) que se ocupa para representar meta-modelos (del nivel 2).

La estrategia en la transformación de modelos tiene su base en establecer primero las relaciones entre meta-modelos para después aplicar estas relaciones (o reglas) a nivel de modelos. En una transformación se dispondrá de un modelo origen, el cual mediante las reglas (a nivel de meta-modelo) llegará a producir una o más transformaciones. El modelo resultante es llamado modelo destino.

En la Figura 2 se presenta esta idea aplicada a las vistas arquitectónicas de software. En ella se aprecian los niveles de modelado. El más alto (M3) es donde está un meta-meta-modelo, porque es donde se encuentra definido el modelo MOF, que sirve para expresar a los meta-modelos del siguiente nivel (M2).

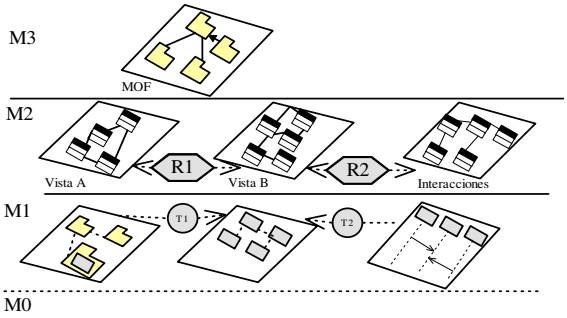


Fig. 2 Transformación de modelos entre vistas arquitectónicas de software.

En este segundo nivel se representan los meta-modelos a relacionar, donde se definen las relaciones de correspondencias. Para el caso de las vistas arquitectónicas se proponen dos tipos de éstas representadas en la figura referida como R1 y R2.

R1 establece las relaciones de correspondencias entre una vista y otra, esto es entre sus meta-modelos (de las vistas A y B). Ésta permitirá llegar a obtener una vista a partir de otra a nivel de modelo, el modelo origen será el *A* y el destino será el *B*.

La otra relación R2, se establece entre el meta-modelo de una vista (B) y el meta-modelo que contiene la información de la *interacción* de los elementos identificados en la fase de requisitos. Esto sirve para que el modelo de la vista que se llegue a obtener (B) se pueda refinar con información adicional (IA) agregándole detalles de dinamismo.

En la parte inferior de esta Figura 2 se indican las transformaciones (T1 y T2) a nivel de modelo, -> indica la dirección en que ésta debe ocurrir.

3.2 Tareas y elementos en una transformación.

Para llevar a cabo las trasformaciones se requiere precisar los elementos de entrada necesarios, y la secuencia a seguir para lograr este propósito. En la Figura 3 se indican éstos.

El primer conjunto de elementos consiste en los meta-modelos antes referidos (los de las vistas A y B, y el de interacciones), éstos se ocupan como entrada para identificar la relaciones entre sus elementos, las cuales después se especifican a través de un lenguaje que permitirá expresar de manera precisa y correcta a estas relaciones. Además éstas deben ser persistentes porque serán requeridas en la transformación de sus modelos tantas veces como se requiera.

La transformación etiquetada con el número 3 ocupa como entrada, además de las relaciones previamente establecidas, al modelo de la vista A, con el cual se obtiene la primera versión del modelo de la vista B, que a su vez será la fuente para la siguiente transformación que también será alimentada con los modelos de iteraciones.

Nótese que tanto el modelo de la vista A como el de las interacciones son extraídos de los requisitos (ésta tarea queda fuera del alcance de este trabajo).

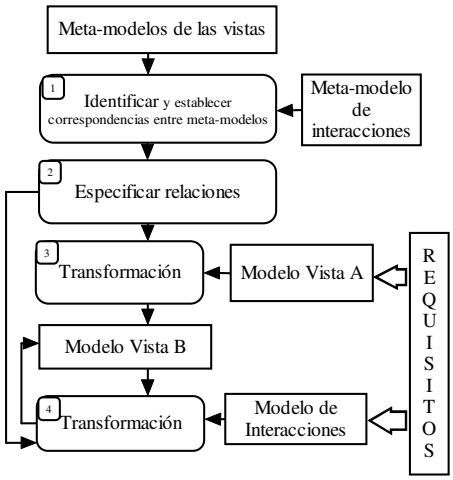


Fig. 3. Elementos requeridos y secuencia de tareas en la transformación de modelos de las vistas.

4. Las vistas arquitectónicas de software y sus meta-modelos.

4.1 Los enfoques de las vistas arquitectónicas.

El concepto de vista tiene muy variadas interpretaciones. El estándar de la IEEE [7], cuya parte se muestra en la Fig. 4, considera a una vista como parte de la arquitectura y la asocia con un punto de vista y con los modelos. Así una vista puede tener muchos, uno o varios modelos para representarla. La situación que se presenta por el lado de los puntos de vista que conforman a una arquitectura es que se abre la posibilidad de tener tantas vistas como puntos de vista se tengan, por lo tanto el problema se traslada a cómo generar sus meta-modelos. No obstante al ser un estándar se debe considerar (aunque sea en términos muy generales) lo que debe tomarse en cuenta en una vista.

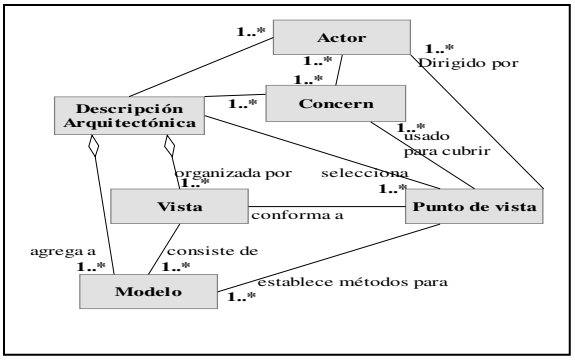


Fig. 4. Meta estructura de la descripción arquitectónica de software, especificado por el estándar 1471 de la IEEE.

Hay otras propuestas más específicas de lo que debe ser una AS. El modelo de 4+1 vistas de [8] considera a cuatro tipos de éstas y el +1 es una vista que relaciona (vincula) a las otras cuatro. En ese enfoque cada vista es una parte del proceso de desarrollo (funcional, de código, de procesos, de despliegue), por tal razón fue adaptada al proceso unificado de desarrollo (RUP), sin embargo estas vistas se tienen que ir construyendo a la vez que se diseña el sistema y la arquitectura pierde la intencionalidad de ser parte de la planeación del diseño. Algo parecido ocurre con la propuesta de [9].

El planteamiento que se hace en [4] y en [10] proponen que las vistas son la representación de las estructuras de software del sistema, lo cual resulta adecuado para diseñarlas antes de detallar al sistema. Así una vista se presenta como una forma de modelar con antelación a un sistema de software.

Por lo tanto este enfoque es el que se seguirá para hacer el modelado de dos vistas, y se seleccionan a la modular y a la de componentes-conectores (C-C) porque representan dos tipos de estructuras que tienen los sistemas de software, la estructura estática y la dinámica. La primera corresponde a la modular, la cual a través de módulos representa las relaciones de los

macro elementos de un sistema. La estructura dinámica es bien representada por la vista de componentes y conectores. En la figura 5 se presenta a estas dos vistas a nivel de modelo, ahí se pueden apreciar sus elementos primarios (módulos y componentes-conectores). Como se indicó en la sección 3.2, para poder llevar a cabo las transformaciones es necesario crear sus meta-modelos.

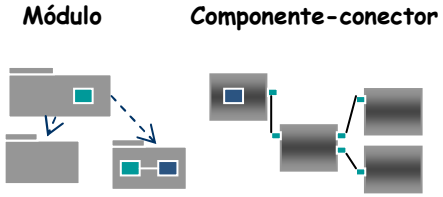
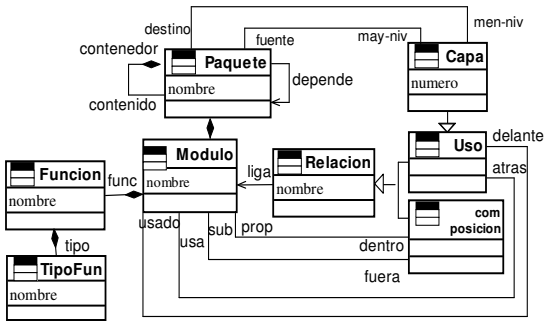


Fig. 5 Modelos de los tipos de vistas: modular y componente conectores

4.2 Meta-modelo para la vista-modular.

El meta-modelo de la vista-modular es mostrado en la figura 6, el cual está conformado por dos elementos primarios que son "Modulo" y "Paquete", el "Modulo" posee una propiedad que es la función que desempeña (Funcion), pudiendo ser más de una, a su vez la función tienen un propiedad de "Tipo", cuyo nombre puede ser: "LOCAL", "SALIDA" o "INTERACCION". Indicando la forma en que la "Funcion" es accedida: "LOCAL" significa que sólo se activa del exterior pero sus efectos son locales; "SALIDA" significa que produce alguna información; y por último "INTERACCION" significa que requiere información de otro "Modulo" y a su vez entrega información.

Se han modelado los estilos arquitectónicos propuestos en [4]: (capa, uso y composición) como tipos de relación que se establece entre los módulos o entre los paquetes y las etiquetas en las ligas de asociación son útiles para establecer las relaciones. Por ejemplo en la relación de "Uso" la etiquetas del lado de "Modulo" indican que un objeto de "Modulo" será el "usado" y otro el que "usa".



Fig,. 6 Meta-modelo de la vista-modular

4.3 Meta-modelo para la vista-C-C.

Este meta-modelo para la vista-C-C es mostrado en la figura 7, ahí se indica que sus elementos primarios son componente y conector, los cuales a su vez son sub-tipos del Tipo-comp, este último es un componente genérico que incluye como atributo el nombre (heredado a sus clases Componente y Conector),

Una clase "Componente" a su vez incluye como atributos a "Servicio" y "Puerto", a su vez un objeto de servicio está asociado con un objeto de Puerto, pues es por donde se establecerá la comunicación de dicho servicio al exterior del componente.

Por el otro lado, el Conector tiene a su vez como atributo al Rol, y debido a que un objeto de la clase Conector vincula a un módulo con otro, se establece una relación binaria de asociación entre Conector y Componente. Por esa misma razón hay una relación de asociación entre el Rol y el Puerto, pues a través de ellos se establece el vínculo de comunicación.

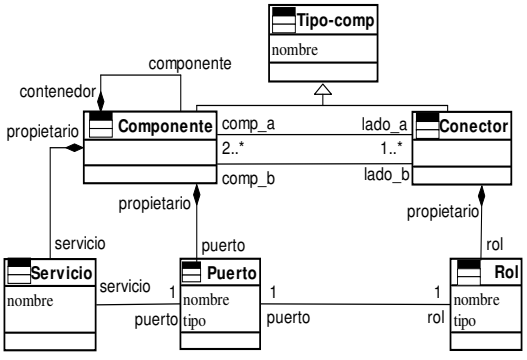


Fig 7. Meta-modelo de la vista-C-C

5. Estableciendo relaciones para la transformación.

5.1 El lenguaje.

Una vez que se tiene a los meta-modelos involucrados en la transformación, el siguiente paso es proceder a establecer las relaciones entre ellos, es decir, especificar las reglas que se seguirán para llevar a cabo una transformación, en este caso la transformación entre las vistas arquitectónicas.

Esto se hace mediante alguno de los lenguajes de transformación de modelos analizados en [11]. Los lenguajes más ocupados son el ATL y el QVT [5], el primero es soportado directamente por el entorno de trabajo de ECLIPSE [12], y el segundo es una propuesta dada por OMG, aquí se usará este último lenguaje ya que tiene como una de sus ventajas el poder expresar las reglas de relación en código y en notación gráfica lo cual permite ilustrar mejor la propuesta.

Algunas de sus características relevantes de éste son: que es un lenguaje declarativo pero también usa sentencias imperativas e incorpora también a cláusulas del OCL (Object Constrain Lenguaje).

Una relación se establece usando la palabra clave "relation" y un nombre para identificarla. Hay dos tipos de relaciones: las que se ejecutan sin ninguna llamada explícita (anteponiendo "top" a "relation") y las que se invocan explícitamente, la cual se incluye en un predicado "where". Cada relación involucra a dos dominios (domain), el origen y el destino. En cada dominio se llevan acciones de sólo verificación o de "enforce", lo primero no crea objetos, en cambio lo segundo se usa para crear elementos, esto es generar el modelo destino. La referencia completa de este lenguaje se encuentra en [6].

5.2 Estableciendo relaciones.

Para establecer las relaciones (reglas de transformación) mediante el QVT se ocupa un proceso heurístico: lo primero que se hace es identificar aquéllas relaciones que deben ser ejecutadas en forma automática, es decir, sin que sean llamadas explícitamente por otra relaciones, luego se establecen dependencias entre ellas, posteriormente se eligen aquéllas que no dependan de otras, las cuales serán relaciones de tipo "top" y las restantes se asocian con quienes las van a invocar (o llamar). Esta invocación se hace desde dentro del predicado "where", lo cual permite establecer una liga o secuencia de ejecución. Dentro de ese predicado también se pueden invocar funciones y usar con cláusulas del OCL.

Es importante hacer notar que no todos los elementos de un meta-modelo origen (o fuente) se relacionan o establecen alguna regla con elementos del meta-modelo destino, pues no siempre son modelos cien por ciento compatibles.

En los modelos aquí considerados, esta situación de "no correspondencia" se presenta en la clase relación "Capa" del meta-modelo de la vista modular, la cual no establece ninguna relación con ninguna clase del meta-modelo C-C. Porque esa propiedad de Capa sólo se presenta en la vista modular [4]. También puede darse el caso de que el modelo destino no se relacione con alguna clase del meta-modelo fuente. Sin embargo esto queda fuera del alcance de este trabajo. Siguiendo este proceso se identificaron las relaciones descritas en la tabla 1. donde hay tres tipos de relación que se activan automáticamente y sólo una (funcionAservicio) que es llamada por alguna otra.

Tabla 1. Relaciones identificadas en los meta-modelos: modular y C-C

Relacion	Ti-po	Clases que participan en la relación	
		Modular	C-C
moduloAcomponente	top	Modulo	Componente
funcionAservicio	-	Modulo, Funcion, Tipo	Componente Servicio Puerto
rUsoModAconector	top	Uso, Modulo, Servicio	Conector, Rol, Componente, Puerto, Servicio
rComposisicionModAcomp	top	Composición Modulo	Componente

Relación: moduloAcomponente

Este vínculo "mapea" cada "Modulo" con un "Componente". Note en el código mostrado en la Figura 7 que el objeto de dominio de Modulo sólo será verificado, en cambio el de dominio destino tiene un tipo "enforce" lo cual hace que cree un objeto de la clase "Componente" asociado con la clase "Modulo".

Por otro lado dentro de "where" existe una llamada a la relación "funcionAservicio" relacionando a un objeto de "Modulo" con uno de "Componente" en la invocación de la función (como paso de parámetros). Su diagrama se muestra en la Figura 8 y su código correspondiente en la Figura 9.

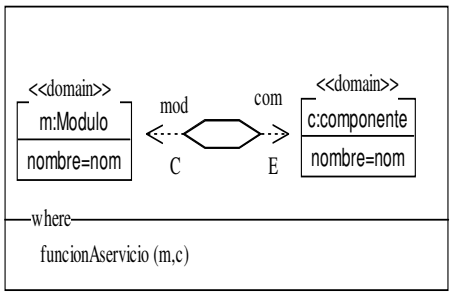


Fig. 8. Relación ModuloAcomponente

```
top relation moduloAComponente
{ nom: String;
  checkonly domain mod m: Modulo {
                                    nombre = nom};
   enforce    domain  com c: Componente{
                                    nombre = nom};
where{ funcionAservicio(m,c)}
}}
```

Fig 9 Relación: ModuloAcomponente (código)

Relación: funcionAservicio.

En este caso esta relación implica que una "Funcion" del meta-modelo "Modulo" generará un "Servicio" de "Componente", al hacerlo también el "Tipo" de la "Funcion" generará un elemento de "Puerto". El "Puerto" tendrá una correspondencia con el nombre del "Tipo" de la siguiente manera: "LOCAL" → "ENTRADA", "SALIDA"→ "SALIDA", "INTERACCION" → "ENT_SAL". Para este propósito se emplea una función de OCL que realizará dicha correspondencia. Note en la cláusula where en la Fig. 10 y 11, cómo la variable tipoPto que será el nombre del puerto, se obtiene llamando a la función tipoPuerto, dicha función tiene código en OCL.

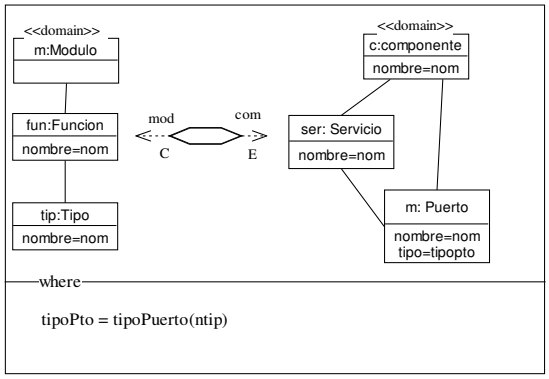


Fig 10. Relación funcionAservicio

Por el lado los meta-modelos fuente se emplean tres clases en esta relación, las cuales sólo serán verificadas al momento de ejecutar las relaciones.

En cambio, las clases los meta-modelos destino serán creadas, aunque solamente las clases "Servicio" y "Puerto", donde éstas establecen una relación con la clase del "Componente" de donde se llamó a este servicio, esto es mostrado en su diagrama de la figura 8 y su código en la figura 9.

```
relation funcionAservicio
{
 nom,ntip,nser,tipoPto:Strings;
 checkonly domain mod m:Modulo {
    funcion = fun:Funcion {nombre=nom,
                tipo = tip:Tipo {nombre=ntip}}}
    enforce domain c:Componente{
                servicio = ser:Servicio {nombre=nser},
                puerto      = pue:Puerto  {nombre=nser +
'pto',
                tipo     = tipoPto}}
    where {tipoPto = tipoPuerto(ntip)
                }
}
 function tipoPuerto(tipoFun:String):String;
 { if (tipoFun = 'LOCAL') then 'ENTRADA'
     else if (tipoFun = 'RESULTADO') then 'SALIDA'
               else                  'ENT_SAL'
                endif
  endif;
 }
}
```

Fig. 11. Relación funcionAservicio (código)

Relación: rUsoModAconector

Aquí se transforma la relación Uso de la meta-clase Modular a un vínculo entre un conector y dos componentes, en la meta-clase C-C, como se ilustra gráficamente en la Fig. 12, la creación de objetos para esta relación es numerosa, puesto que se genera una relación de dos componentes con un conector con sus respectivos puertos. En la Fig 13 se muestra parte del código para destacar cómo dentro del where se llaman a las relaciones que crean la relaciones entre módulo componente y funcionAservicio, las primeras dos crean los componentes a conectar y la segunda sus servicios correspondientes**.** Además se aprecia cómo las variables de la clase Componente serán usadas como parámetros en la relación contenida que dentro de la cláusula *where*.

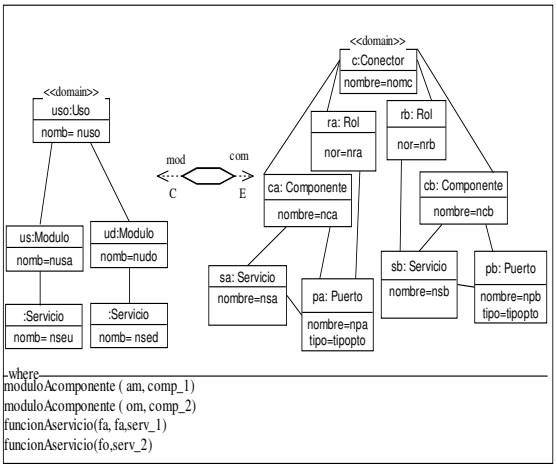


Fig. 12 Relación rUsoModAconector

```
top relation rusoModAConector
  {
     ser_1, ser_2  :Servicio;
     comp_1, comp2:Componente;
     nma,nmo,ncon,nfa,nfo,tfa,tfo : String.
     checkonly domain reluso ru:Uso {nombre=nuso,
     usa = am:Modulo {nombre,nma, fun =
fa:Funcion {nombre=nfa, tipo=tfa}}
           usado = om:Modulo{nombre=nmo,fun =
fo:Funcion {nombre=nfo, tipo=tfo}}
   }.....
.....
……
   where { moduloAComponente (am,comp_1);
           moduloAComponente (om,comp_2);
           funcionAservicio (fa,serv_1);
           funcionAservicio (fb,serv_2);
          }
```

Fig. 13 Relación rUsoModAconector (código)

Relación rComposicionModAcomp.

En este caso la composición de Modulos generará también una composición de componentes, note que al momento de crearse un componente dentro de él se crea el subordinado, de esta manera se establece también una composición de componentes. Ver figuras 12 y 13.

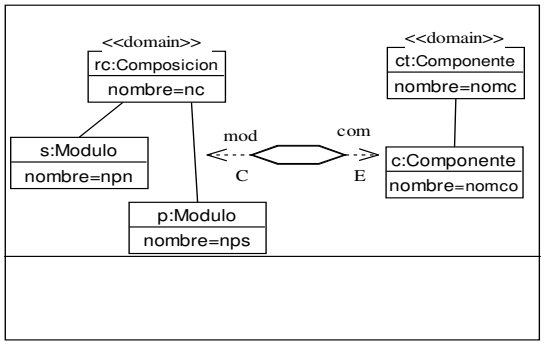


Fig 14. Relación rComposicionModAcomp

```
top relation rcomposicionModAcomp
  { nc, nct, nco, nms, ncomp : String;
    checkonly domain rcomp rc:Composicion
{nombre=nc,
                  prop = mp:Modulo {nombre= nmp}
                  sub = ms:Modulo {nombre=nms}
    }
 enforce domain cc: Componente{nombre =  nct
contenedor  = cc;
  componente  =  c : Componente{ nombre = nco
  }
}
```

Fig 15. Relación rComposicionModAcomp (código)

6. Aplicando la transformación entre vistas a nivel de modelo.

Se plantea como caso de estudio el diseño de una arquitectura de software para un sistema de reservación de boletos (tickets o billetes) para el transporte de pasajeros (avión, autobús, tren). Los requisitos ya fueron analizados y se ocupó el método propuesto en [14], el cual sigue un proceso iterativo y descendente para diseñar una vista. En este caso se generó sólo el modelo de la vista-componente, cuya parte de ésta se muestra en la Fig. 16 ocupando la notación de UML 2.0.

Existen cuatro módulos vinculados entre sí por una relación de uso. *QuerForm*, tiene como responsabilidad generar las interfaces para proporcionar los datos que se van a consultar sobre las distintas líneas de transporte, ahí se debe proporcionar la información necesaria (origen, destino, número de personas, fechas de salida y regreso). Además se mostrarán los distintos itinerarios disponibles que serán generados por el módulo *GeneItin*

El Módulo *QuerForm* será usado por el módulo *Parchase* que tendrá como responsabilidad llevar a cabo las funciones relacionadas con el pago de los boletos. Este último módulo usará al de reservación donde se supone se asigna el o los asientos correspondientes al tipo de medio de transporte.

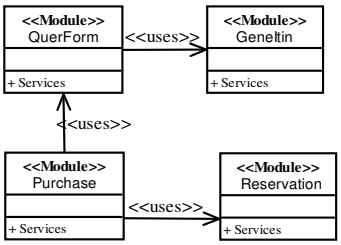


Fig 16. Modelo de una vista modular.

Ahora se aplican las reglas que se han establecido a nivel de meta-modelo, para lo cual se ocupó la propuesta de MOMENT [15] que usa como marco de trabajo a ECLIPSE, tanto para la representación de los meta-modelos y de los modelos, como para la ejecución de la transformación

Las reglas o tipo de relaciones que fueron usadas en este caso son las de ModuloAcomponente y la de "rUsoModAConector" puesto que sólo hay relaciones de <<uso>> en el modelo de la vista modular, además por supuesto de los módulos. La aplicación de estas reglas llega a producir la primera versión del modelo de una vista arquitectónica de C-C, tal como el que se muestra en la figura 17, donde Session y Access son de clase "Conector".

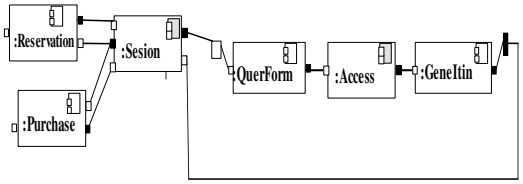


Fig 17. Modelo generado de una vista C-C.

Esta primera versión del modelo de la vista C-C puede ser refinada al aplicarle una transformación usando el modelo de escenarios. En la figura 17 se muestra una iteración entre QuerForm y GeneItin, donde se percibe cómo el elemento GenItin es descompuesto en dos: Itinerari y Routes, de tal manera que si se aplica de nuevo la transformación, el modelo de la vista reflejará este refinamiento tal como se muestra en la figura 19.

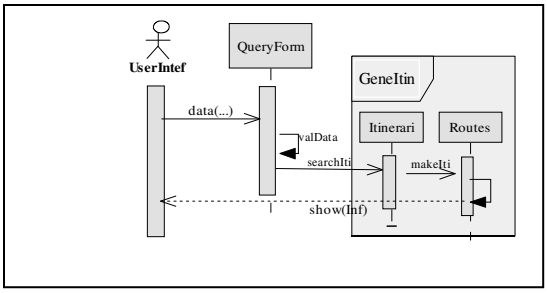


Fig. 18. Modelo de Escenarios usando diagrama de secuencias en UML 2.0

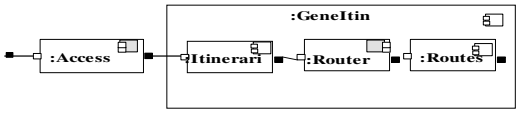


Fig. 19. Refinamiento del modelo de la vista C-C.

7. Conclusiones

Al haber establecido un vínculo entre el MDA y el AS se muestra cómo es posible diseñar arquitecturas de software con más rapidez y con re-uso mediante orientación por modelo. Con la creación de los meta-modelos arquitectónicos y de las reglas de transformación se ha mostrado la forma de modelar y diseñar arquitecturas de software aplicable inclusive para otros tipos de vistas. El generar más rápidamente la vista modular y la de C-C a nivel de modelo para un sistema, hará posible que más sistemas cuenten con una arquitectura más integral (de dos vistas). Tanto los meta-modelos como las relaciones realizadas son extensibles, esto es, pueden incorporarse más relaciones enriqueciendo las transformaciones. La heurística planteada para llevar a cabo el proceso de transformación, aunque informal**,** servirá como un primer paso para crear un método de diseño que permita hacer en un futuro cercano todo el proceso automáticamente, aunque es claro que la identificación de relaciones sigue siendo uno de los más grandes retos en el proceso de modelado.

Se reconoce que el camino hacia una completa automatización con el modelado aún es largo pero se ha dado un primer paso en relacionar estas dos novedosas áreas de la ingeniería de software

.

Referencias

[1] Arlow J., Neustadt I, "Enterprise Patterns and MDA: Building Better Software with Archetype Patterns and UML, Addison Wesley Professional, ISBN: 978-0-321-11230-9, 2003.

[2] OMG, Object Management Group: Model Driven Architecture (MDA). 2004. http://www.omg.org/cgi-bin/doc?formal/03-06-01

[3] Shaw and Garlan, Software Architecture: perspectives on an emerging discipline, Prentice Hall ISBN 0-13-182957-2, 1996

[4] Bass L., Clements P., Kazman R., Software architecture in practice, Addison-Wesley, 2ª Ed. , ISBN 0-321-14595-9, 2003.

[5] IEEE Computer, Vol. 38, N. 2. Feb. 2006.

[6] IEEE Software, Volume: 23 Mar-Apr 2006.

[7] IEEE Product No. SH94869-TBR, IEEE Recommended Practice for Architectural Description of Software-Intensive Systems, 2000

[8] Kruchten Philippe. The 4+1 View Model of Architecture, Paper published in IEEE Software Vol.12, No.6, pp. 42-50 Nov. 1995.

[9] .Hofmeister, C., Robert Nord, and Dilip Soni. Applied Software Architecture. Boston, MA: Addison Wesley, 2000

[10] Clements P., Bachmann F., Bass L., Garlan D., Ivers J., Little R., Nord R. Stafford J. Documenting Software Architecture, Views and Beyond, Addison-Wesley, 2002.

[11] Krzysztof Czarnecki and Simon Helsen, Classification of Model Transformation Approaches OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture, 2003.

[12] ECLIPSE: http://www.eclipse.org/

[13] OMG 2nd Revised Submission: MOF 2.0 Query/Views/Transformations. http://www.omg.org/cgi-bin/doc?ad/05-03-02

[14] Limon C. R., Ramos S.I., Torres J. J. Designing Aspectual Architecture Views in Aspect-Oriented Software Development , Computational Science and Its Applications, LNCS 3983, pp, 726-735, 2006.

[15] El proyecto MOMENT: http://moment.dsic.upv.es/

Analyzing and Designing Software Architecture Views driven by their Relationships

Rogelio Limon Cordero[1], Isidro Ramos Salavert[1],
Maricela Morales Hernández[2], Jorge Zaráte Perez[2]
[1]*Department of Information Systems and Computation, Technical University of Valencia,*
Camino de Vera s/n E-46022 Valencia –
{rlimon,iramos}@dsic.upv.es
[2]*Departamento de Sistemas y Computación, Instituto Tecnológico de Oaxaca*
Av. Victor B. Ahuja 125 esq. Calz. Tecnológico, Oaxaca, México
{mmorales, jorge.zarate}@itoaxaca.edu.mx

Abstract

Software architecture views represent the basic structures of a complex software system. By means of these views, it is possible to shape the different concerns that appear in the requirements and design phases. Each view specifies the elements that must be built up in the detailed design, and the relationships that must be established among them. This paper makes an analysis of the relationships present in two view types (modular and component-connector views). This work establishes how these relationships can be shaped by means of theirs styles in an operative way. This approach allows a better architectural design through the management of the relationships, using their styles as the basis for the architectural structures and leads to a more precise analysis in the design for architectural views.

1. Introduction

Software architecture is a key element in software development because it is the bridge between the requirements and design phases. Its design allows the user to determine the structures that will shape the software system and the way in which their elements communicate, before proceeding to their detailed design. Each structure is really an architectural view, which contains architectural element relationships with each other.

Architectural view design implies identifying and specifying the elements and establishing their relationships. This last task is perhaps one of the most important because the relationships constitute the basis of the architectural structure.

The design methods used in [1][2][3] establish architectural relationships by using a set of prototypical structures called architectural patterns as a reference, also known in other works as architectural styles. However, no analysis of these styles has been made in these methods nor have they been used in an operational way in the design process.

The aim of this paper is to research this area by considering the architectural style as a relationship instead of a type of view. In addition, a proposal is presented to model the relationships in order to make the styles operative in the design and analysis phases of the modular and component-connector architectural views, with all of this being driven by an iterative and descending method presented in [4].

This paper is structured as follows: Section 2 proposes meta-models for the types of modular and component-connector views to establish the relation between the styles and the other architectural elements. Section 3 describes the types of architectural relationships, which are represented with matrices and explains how the proposed mechanism operates. Section 4 proposes an adaptation of these matrices to the architectural styles. Section 5 proposes a quantitative analysis of the architectural relationships. Section 6 shows how the architectural styles are applied in the design method, which is illustrated in a case study. Finally, the conclusions are presented.

2 Architectural Views and their Styles

From the definition presented in [1], the architectural view is considered as a specific perspective of each structure of the architecture, whose formation rules are contained in a viewpoint [5]. We will use the term view-type as the specification that defines the elements that an architectural view contains and the type of relations that can be established among them.

There are several approaches about what the architectural view should be; among the more relevant, we can enumerate the following: SEI[1], SIEMES[2], Kruchten [6]. Even though there are some differences among them about how they interpret a view, there are coincidences in the type of elements and relations that are considered in each one. In all of these approaches, the components and connectors are considered as basic building blocks in some of their views, and the module element is included in different ways. The focus in this work is to analyze and deal with the relationships of elements of this kind. Therefore, the component-connector(C-C) and modular view-types [1] have been chosen as the most appropriate approaches to apply to our proposal. Furthermore these approaches contain the two types of relationships present in almost all software architectures of systems, dynamic and static relationships. The C-C view-type basically includes communication relationships (dynamic), and the modular view-type includes only structural relationships (static).

The relationships among the elements of a view-type adopt either some of the pre-established shapes known as architectural styles (or simply styles) [6] or similar shapes. Styles define a set of features and constraints that typify the relationship among the architectural elements. In this paper, the term basic style is used to denote a specific and well-delimited style, and the term heterogeneous style is used to denote when a style is composed of two or more basic styles.

The basic styles used in software architecture are studied in [7],[8]. In [7], each style is associated to a specific type of view, which means that an architectural style cannot be inside of more than one type of view. This is derived from the fact that each type of view deals with different types of concerns [5], and a specific style is used for shaping each type of concern.

Figure 1 shows the two meta-models proposed. Figure 1 (a) depicts the meta-model for designing a modular view type (<<Modular-VT>>), its elements are <<Modules>>, which are linked with each other according to three types of styles: <<Decomposition>>, <<Uses>> and <<Layered>>. The links established among the elements are relationships of dependence.

Figure 1 (b) depicts the meta-model for designing a component-connector view type (<<C-C-VT>>) specifying its relation with styles. This meta-model shows the elements and relationships that make up this type of view. The primary elements are <<Component>> and <<Connector>>, which are linked by their <<Ports>>. The <<Ports>> can contain <<Interfaces>> to allow several modes of access. Both the components and connectors are determined by the style types: <<Pipe&Filter>>, <<ClientServer>>, <<PeerToPeer>>, <<PublishSubscribe>>. Also, other styles can be included (<<OtherStyle>>).

Note that, as Figure 1 shows, the composition relationship in a <<ModStyle>> and in a <<C-CStyle>>, allows the integration of self-contained styles in both the (a) and (b) meta-models, which makes it possible to build heterogeneous styles.

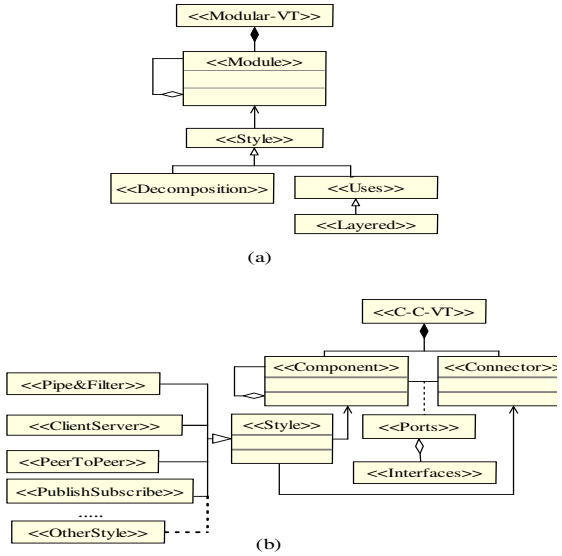


Fig.1 Meta-model for: (a) The modular view type, and (b) The C-C view type.

3. Establishing the Basic Architectural Relationships

Considering that an architectural style is a specific type of a relationship that is established between at least two architectural elements, a mechanism must be established to identify, represent, and analyze such relationships in order to improve the architectural design process and to achieve a quantitative analysis.

In this paper, we propose a strategy that creates a basis for constituting each style from the relationship between two elements.

(i)

↓→	A	B
A	0	*1*
B	*0*	0

(ii)

↓→	A	B
A	0	*0*
B	*1*	0

(iii)

↓→	A	B
A	0	*1*
B	*1*	0

Figure 3. Relationship cases with R=2

Our approach is based on the design rules of [9], where a dependency matrix for analyzing relations among elements is proposed. This matrix has been used for the hierarchical and dependency relationships [10]. In this paper, a basis binary relationship [11] is used to represent the relationships in each one of the styles of the component-connector and modular view-types. This is done to be able to do the following: a) model the communication relationships among components, b) model the composition relationship among components and modules, c) model the use and layer relationships, and d) represent these relationships with matrices to obtain the architectural model.

3.1 Basic Relationship

Let's consider that a relationship between two elements (components or modules) named A and B can be represented by a relationship matrix as shown in Figure 2.

↓→	A	B
A	0	V_x
B	V_y	0

Figure 2. Relationship matrix between two elements (binary relation)

In the relationship matrix, Vx, and Vy (in a cell) represent values that indicate the relation between their corresponding row and column elements. Each value has a different interpretation to indicate the type of relationship. Using a representation of binary relational algebra [11], A Vx B means that A and B have a type of relationship that is represented by the value of Vx, where Vx and Vy are associated with a type of relationship (A_Relationship and B_Relationship) by means of a correspondence, i.e. types of relationships = {(0, null), (Vx, A_Relationship), (Vy, B_Relationship)}

Note that the matrix does not include self-relationships (reflexive) among the elements, which is indicated by the value 0 in their corresponding cells.

The range of values for Vx and Vy varies from 0 to R-1, where R is the number associated to the relationship types, and the ´-1` is due to the fact that the null relationship (0) is included in R.

In the matrix, R2–1 relationship cases between two elements can be generated (the non relation case is excluded). For instance, if two elements (A and B) can be related in only one way, then R=2 (null included), and the range of values for Vx and Vy will be {0,1}. Thus, 2 different relationship cases can be established between A and B. Figure 3 are shows all the possible cases with these values.

If the value 1 is associated with a dependence relationship, then each case is read as: (i) A is dependent on B, (ii) B is dependent on A, and (iii) A and B are mutually dependent. Note that the cases (i) and (ii) represent the same relationship type (simple dependent) because this relationship is symmetrical; on the other hand (iii) represents a different type of relationship (interdependence).

The following annotations complement this proposal: a) R depends on the number of styles that are considered in each type of view. b) Each matrix is used to analysis and build relationships, and to maintain these relationships for evolution and maintenance.

3.2 Communication Relationships

Communication relationships are given in the component-connector view-type, where a component communicates with another component by means of a connector. The components have a specific computational behavior, such as client, server, filter, object, or database. The connectors assume the role of coordinating and controlling the communication between two components, or among several components. Therefore, the connectors are in charge of the way communication is done. The interactions that are attributable to the connectors are the following: streams, data access, procedure calls to invoke services, send of messages, and events. These are described in [8].

Types of communication relationships are established according to what it is communicated and how it is dealt with. There are two basic sets of these relationships: a) communication of data (DC), similar to streams, and data access; and b) sending of control information (CC), similar to procedure calls, sending of messages, and events. Different type of communications can be formed using these two kinds of relationship by means of the relationship matrix.

The interpretation of a communication relationship using the relationship matrix is as follows: (see Figure 2) starting from the first row, the component of each

row (A or B) communicates something to the component of its corresponding column (A or B). The values Vx, and Vy in one cell are used to specify the type of communication that will be carried out by the connectors.

The values for generating the relationship matrices for the two types of communication described above are: R= 3; Vx, Vy = {0, 1, 2}, where their correspondences are: {(0, null), (1, DC), (2, CC)}. There are eight relationships that can be generated from these two basic relationships.

Figure 4 shows the possible cases that can be built with R=3, the cases presented in Figure 3 are not included. The following notation is used to clarify the communication relationships: → indicates communication of data in one direction; ↔ indicates communication of data in two directions; => indicates control communication in one direction; and <=> indicates control communication in two directions.

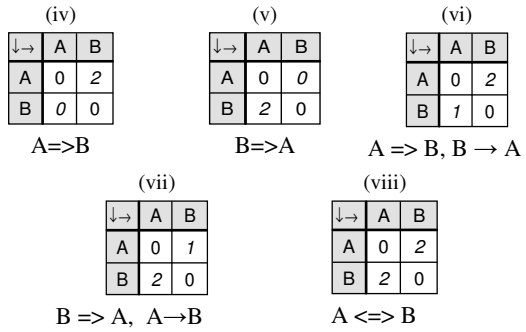


Figure 4. Relationship cases with R=3

Note that only five of the eight cases are different (see Figure 3 and Figure 4). Case (ii) is symmetrical with the case (i). The same happens with cases (iv) and (v), and with cases (vi) and (vii). The (i) and (iii) cases indicate that there is a communication of data from A component to B component, furthermore in case (iii) there is another communication from B to A. The cases (iv) and (viii) indicate that there is a control communication from A to B, furthermore in case (viii) there is another communication from B to A. The case (vi) combines a data communication with one control communication.

3.3 Composition Relationship

To accomplish the relations described in the meta-models of the modular and component-connector views (shown in Figure 1), the relationships among the components or among modules are called composition relationships, which must be taken into account. This type of relationship allows us to combine several basic styles in order to create a heterogeneous style. Therefore, the relationship matrix must be adapted for this purpose.

When an element (module or component) contains other elements, the corresponding rows and columns of the matrix relationship are divided according to the number of elements. These must be clustered in a container element which is shown in Figure 5(a). A specific case with k = 3 is shown in Figure 5 (b). It represents a relationship that is divided into sub-elements of the "D" element, where the D element has a relationship with the "A" element (Figure 5 (c)). This establishes the relations among sub-elements and their relation to the other elements.

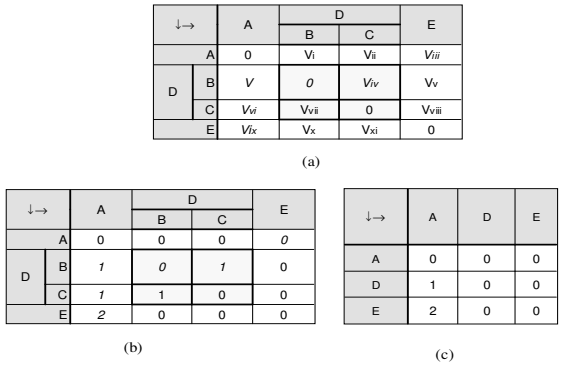


Figure 5. Composition relationship: (a) general, (b) a specific example for k=3, (c) hiding the sub-elements

3.4 Dependence Relationship

This type of relationship is only applied in modular view-types, because it corresponds to the links that are established when an architecture is divided into elements called modules. In this case, a module can be dependent on another module (or not be dependent on any module), because it requires using the services of another module. If both linked modules are in an even plane (layer) this relationship is called <<uses>>; however if these modules are in different planes its relationship is called <<layered>> (see Figure 1). Note that in this type of relationship, the flow of communication does not matter because their relationships only indicate how the elements are related structurally.

The relationship matrix for representing the dependence relationships is defined with R=2. Its possible cases are shown in Figure 3, where the value 0 indicates no dependence, and the value 1 indicates that there is a dependence. The matrix for composition

relationships is also used when a module includes others modules (<<decomposition>> style). However for a <<layered>> style it is necessary to specify the layer where a module is locted. For this reason, the number of layers is included in the upper left corner of the relationship matrix as shown in Figure 6. One column is added to indicate which modules of the different layers are linked.

Num. Layer		IdEle	A	D		E
				B	C	
	A	X	0	v	v	v
D	B	–	v	*0*	*v*	v
	C	–	v	v	0	v
	E	*Y*	v	v	v	0

Where: NumLayer = 0,1,2… V = {0, 1}
IdEle = Identification (literal or number) of a element of a matrix of different layer

Figure 6. Dependence relationship matrix (with sub-elements)

4. From the Relationship Matrix to the Architectural Style

4.1 Architectural Style Representations for the Component-Connector View-type

The component-connector view shows the dynamism of software architecture, therefore the communication links are established in the relationship between their elements. These links are typified by the architectural styles belonging to this view-type. The styles set the standard of the element relationships, and determine the behavior of the elements.

Each style is associated to a relationship matrix, where its elements are the components and where the cells of this relationship matrix are the connectors of the class that correspond to its style. The architecture is represented in UML 2.0..The basic notation employed here is shown in Figure 7.

The architectural styles included in Figure 1 (b) are represented by means of matrices and their corresponding UML representation, as shown in Figure 8.

In this case a matrix relationship with R=3 is used. This value makes it possible to generate the communication relationships of the styles considered in Figure (1) b).

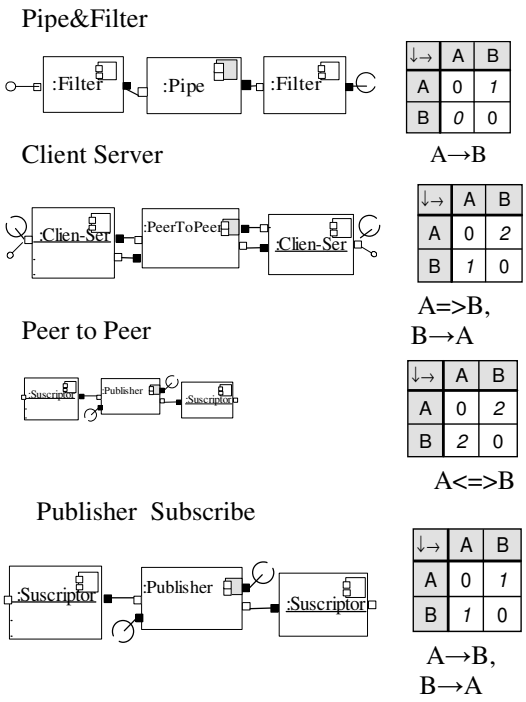
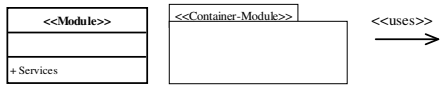


Figure 8. Matrices for basic architectural styles of the component-connector view and their corresponding UML representation

4.2 Architectural Style Representations for the Modular View

In this case, the dependence and composition relationships are used because the modular view only represents a static relationship; therefore, the value of R for the matrix is 2. The corresponding UML 2.0 notation for this view is shown in Figure 9.

The architectural styles included in Figure 1(a) are represented by means of matrices and their corresponding UML representation, as shown in Figure 9.



Key: UML 2.0

Figure 9. Basic notation for the modular view representation

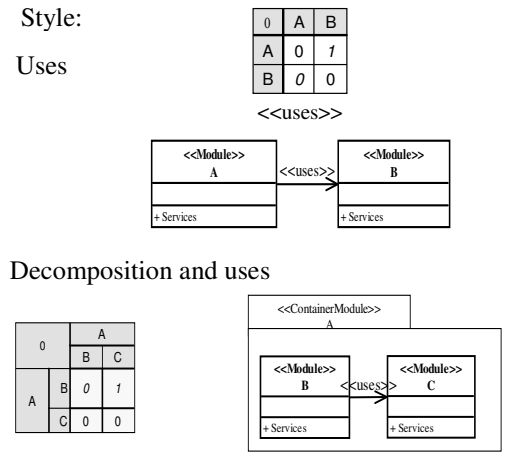


Figure 10. Matrices for basic architectural styles of a modular view, and their corresponding representation

5. Architectural styles Analysis

There are several methods for analyzing software architectures. All of them have as a main goal to assess quality attributes of the system, such as availability, security, modifiability, testability, manageability, performance, usability and son on. This is done using different techniques and different criteria for assessing one or several of these attributes. In a survey that was done on eight methods [12], it was shown how most of these techniques use qualitative criteria to assess the quality attributes of the system; for example ATAM [1][12], which is currently widely used. Very few methods use quantitative criteria to assess the quality attributes. It is import to note that in all methods the intervention of stakeholders in necessary for making decisions.

The goal in this work is not to propose a new method of analysis, but to improve the current methods through of the inclusion indicators that evaluate the relationship types used in the software architecture views. This indicators use the relationship matrices as objects of analysis. By means of these matrices we can determine how the architectural styles are used and how they affect some of the quality attributes of the system. Only a few indicators are included in this work to illustrate how they can be evaluated using relationship matrices.

The indicators quantify the architectural styles used in each architectural view, taking into account the relationship types that their styles have.

In a modular view-type the styles link the modules (<<Module>> and <<ContainerModule>>) through the *<<uses>>, <<layered >>, and <<decomposition>> relationships. The interest in these relationships is focused on how the modules are distributed, i.e. if a <<module>> is overused by others, or if a <<Cointainer-Module>> includes too many elements. The following indicators are identified to assess this:*

$UM_{i,m}$: means the module m is used by the module i.

LD_m : Level of dependence in a module *m*, is the number of modules that use module *m*.

DX : Maximum level of dependence in the view, i.e. $MAX(LD_m)$

IM_m : Number of modules included in the module *m*

TM : Total of container modules in the view

AM : Average number of modules included in the modular view.

MD_m : Modular density of module *m*, where *m* is a module of the <<container>> class.

The valuations in some indicators are done:

$LD_m = \Sigma_{i=1} UM_{i,m}$ where $i \neq m$

$MD_m = NM_m / AM$; where $AM = \Sigma_{i=1} NM_i / TM$

These can be evaluated using the relationship matrix of the view to be analyzed. For example UMi,m is obtained in a straightforward way accessing row i and column m from the matrix. LDm for the element m is calculated by summing all the values of each row of the column where element m is located, and so on for other indicators.

These indicators are useful for making comparisons between two or more views generated in the design process, or for restructuring a module when its MD > 1 or when it has a high value of LD.

A similar analysis is made for the C-C View-type, which is oriented to the communication relationship, however this is not included for reasons of brevity.

6. Applying the Styles in the Design of the Architectural View

6.1 Using Architectural Styles in the View Design Method.

Designing a view-type means applying a method to constitute its elements and the architectural

relationships, which, in this paper, are expressed in UML 2.0.

The design method proposed in [4] is used. The activities that are related to our proposal are shown in Figure 11. Our attention is focused on: "Shaping architectural relationships" which is the key to improving the design process by means of style relationships.

The starting point is the initial architecture derived from the functional requirements (expressed by the use cases) and the modules to be decomposed (using the specification of each use case). The found modules are linked or decomposed according to their architectural style; next, a component is derived from each module so the task now consists in finding the relation among the components that will constitute the architecture. The architectural relationships that are established with the architectural basic styles are applied for each architectural view. Their activities are detailed in Figure 11 (b).

The first activity consists of identifying a pair of elements (modules or components) that interact with each other by using the uses cases and scenarios as reference. The interaction is recorded in the relationship matrix, using the corresponding style. In the case of a first time link of these elements, a basic style will be constituted; otherwise, a heterogeneous style will be made, which implies a style reconstruction. Once that style is typified, the architectural representation is carried out.

The activities described above are repeated until there are no elements left to relate. These activities are illustrated by means of a case study in the following section.

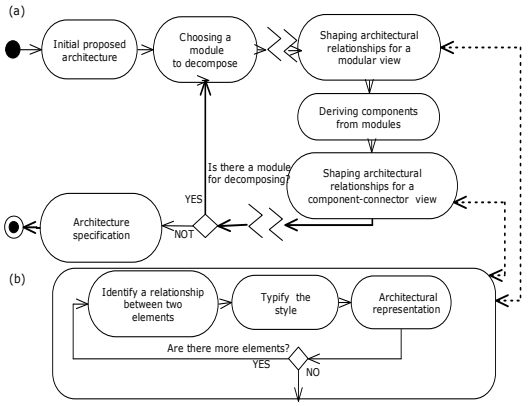


Figure. 11. (a) The architectural design method activities, (partially), (b) The activities to shape architectural relationships

6.2 Case Study

The case study is an extension of the case presented in [4]. This study consists of designing an architecture for a purchase and reservation ticket system for any kind of passenger transport (bus, air plane, train, etc), where the passengers can buy and reserve a ticket, or just search the Internet. After introducing the query, the system will generate a tentative itinerary as a first step.

With the intention to focus on the use of architecture styles proposed here, let's suppose the detection of elements (modules) in a first architecture derived from the use cases, which are shown in Table 1 ; and their scenarios are shown in the sequence diagram in Figure 12.

Table 1 Modules detected

Description	Main Modules
Query formation	QuerForm
Generating itinerary alternatives and routes	GeneItin
Making the reservation	Reservation
Purchasing a ticket	Purchase

Using the scenarios as the source, the type of modules to consider in the modular view and the relationships among them are detected. With this information the relationship matrices are shaped. Then the corresponding architectural representation for this view is built up, following the activities depicted in Figure 11(b). The matrix and representation for this view are shown in Figure 13.

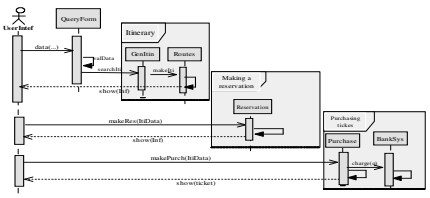


Figure 12. Sequence diagram for purchase and reservation of tickets

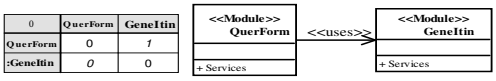


Figure 13. Matrix for the QueryForm and GeneItin modules and its architectural representation.

Once a partial creation of the modular architectonic view is done, their corresponding component-connector view must begin to be shaped. The modular architectonic view is used as a start point to begin the shaping of the component-connector view. Therefore, the first C-C view components becomes from the modular view modules. In the study case, the identification name of each module is then provided as a identification for each component of the C-C view-type.

Then, the information of the scenarios is again used to identify interactions among the components. The first identified interaction appears between the *QueryForm* and *GeneItin* components, where the former transmits the information of a query to the latter, which also generates alternative routes. In this case, the relationship matrix and its architectural representation are shown in Figure 14.

↓→	QuerForm	GeneItin
QuerForm	0	*1*
GeneItin	*0*	0

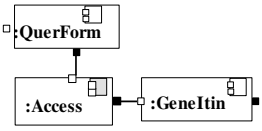


Figure 14. Matrix for the QuerForm and GeneItin components and its architectural representation (Pipe&Filter Style)

This process continues in an iterative way as depicted in Figure 10, until both views are accomplished. The terminal relationship matrices and their corresponding representation for each view are shown in Figure 15 and Figure 16.

(a)

0		Reservation	Make-Itin		Purchase
			QuerForm	GeneItin	
Reservation		0	1	1	*0*
Make-Itin	QuerForm	*0*	*0*	*1*	0
	GeneItin	*0*	0	0	0
Purchase		*1*	1	1	0

(b)

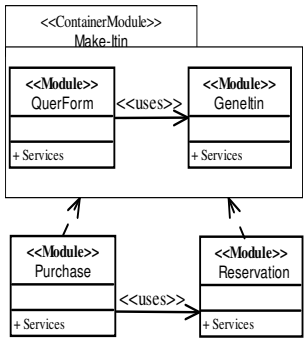


Figure 15. The modular view for the ticket purchase-reservation system, (a) its relationship matrix, and (b) its architectural representation,

(a)

Ticket Reservation-Purchase		Reservation	Make-Itin		Purchase
			QuerForm	GeneItin	
Reservation		0	2	0	*0*
Make-Itin	QuerForm	*0*	*0*	*1*	0
	GeneItin	*1*	0	0	1
Purchase		*0*	2	0	0

Ticket-Reservation-Purchase	Reservation	MakeItin	Purchase
Reservation	0	2	0
MakeItin +	1	0	1
Purchase	0	2	0

(b)

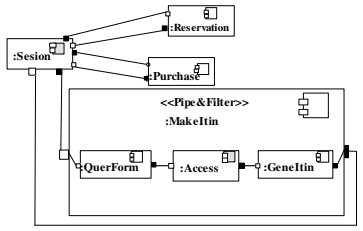


Fig 16 The component-connector view for the ticket purchase-reservation system, (a) its relationship matrix, and (b) its architectural representation

7 Conclusions

By modeling the architectural style as a type of relationship, we have proposed a novel way to design an architectural view. The identification and analysis of the basic relationships proposed here can be used in other view types.

The use of the relationship matrices allows us to: a) adapt some view types where there is dynamism among the element relationships; b) represent structural relationships as composition of architectural views; c) make a quantitative analysis using relationship matrices.

d) make the maintenance of the architecture easier since the element relationships are recorded (this is not addressed in this paper); e) generate a graphical architectural representation from the relationship matrix (in UML 2.0 notation).

Finally, it is shown that the application of the architectural styles improves the method of design of the software architectural views by making the analysis of relations in the architectural design process operational.

References

1. Bass L., Clements P., Kazman R., Software architecture in practice, Addison-Wesley, 2ª Ed. , ISBN 0-321-14595-9, 2003.
2. Hofmeister, C., Robert Nord, and Dilip Soni. Applied Software Architecture. Boston, MA: Addison Wesley, 2000.
3. Rozanski Nick, Woods Eoin, Software Systems Architecture, Working With Stakeholders Using Viewpoints and Perspectives, Addison-Wesley, ISBN 0-321-11229-6, 2005
4. Limon C. R., Ramos S.I., Torres J. J. Designing Aspectual Architecture Views in Aspect-Oriented Software Development , Computational Science and Its Applications, LNCS 3983, pp, 726-735, 2006
5. IEEE Architecture Working Group, IEEE Recommended Practice for Architectural Description of Software-Intensive Systems, 2000.
6. Kruchten Philippe. The 4+1 View Model of Architecture, Paper published in IEEE Software Vol.12, No.6, pp. 42-50 Nov. 1995.
7. Clements P., Bachmann F., Bass L., Garlan D., Ivers J., Little R., Nord R. Stafford J. Documenting Software Architecture, Views and Beyond, Addison-Wesley, 2002.
8. Mehta N. R., Medvidovic N., Phadke S., Towards a Taxonomy of Software Connectors, Proceedings of the International Conference on Software Engineering- ICSE'00 (2000).
9. Baldwin, C.,Clark, K. Design Rules: Volume 1. The Power of Modularity. Cambridge, MA: The MIT Press. 2000.
10. Sangal N., Jordan, E. Sinha, V., Jackson D. Using Dependency Models to Manage Complex Software Architecture, Proceedings of the 20th annual ACM SIGPLAN Pages: 167 – 176, 2005.
11. R. C. Holt, Binary Relational Algebra Applied to Software Architectural, CSRI Technical Report 345, 1996
12. Dobrica L. and Niemelä E. , A survey on software architecture analysis methods, IEEE Transactions On Software Engineering, Vol. 28, No. 7, pp. 638-653, 2002

Aplicando MDA al Diseño de un Datawarehouse Temporal

Carlos G. Neil
Universidad Abierta Interamericana
Facultad de Tecnología Informática
Buenos Aires, Argentina
carlos.neil@vaneduc.edu.ar

Claudia F. Pons
Universidad Abierta Interamericana
Facultad de Tecnología Informática
CONICET
Buenos Aires, Argentina
cpons@info.unlp.edu.ar

Resumen

Model-Driven Architecture (MDA) es un enfoque ampliamente aceptado para el desarrollo de sistemas de software complejos. MDA propone el uso de modelos en todas las fases de desarrollo, desde la especificación y análisis hasta la implementación. La transformación de modelos es la base de MDA; comenzando por un modelo independiente de la plataforma el objetivo es lograr, en cada paso, modelos más específicos. Adhiriendo a la filosofía MDA, en este artículo, presentamos una metodología para el diseño de un datawarehouse temporal que permite definir los conceptos independientes de la implementación. Nuestro propósito consiste, aplicando el enfoque MDA, en la definición de metamodelos y reglas de transformación formales que provean un marco para el refinamiento de un modelo de datos temporal para la obtención de un esquema relacional.

1. Introduction

Las empresas utilizan los datos acumulados durante años, empleados en las transacciones comerciales, para ayudar a comprender y dirigir sus negocios; con ese propósito, los datos de las diferentes actividades se almacenan y consolidan en una base de datos central denominada datawarehouse; los analistas lo utilizan para extraer información de sus negocios que les permita tomar mejores decisiones [12]. Un datawarehouse es una colección de datos no volátiles, que se acumulan en el tiempo, que están orientados a un tema determinado y que se utilizan para tomar decisiones organizacionales [13]. El modelo multidimensional constituye la base del datawarehouse, en él la información se estructura en hechos y dimensiones; un hecho es un tema de interés para la empresa, se describe mediante atributos denominados atributos de hecho, éstos están contenidos en celdas o puntos en el cubo de datos. Un cubo de datos es una representación multidimensional de datos donde éstos pueden verse desde distintos puntos de vista; está formado por dimensiones que determinan la granularidad para la representación de hechos y jerarquías que muestran cómo las instancias de hechos pueden ser agrupadas y seleccionadas para los procesos de toma de decisión [3].

En el datawarehouse el tiempo es una de las dimensiones para el análisis [8], [9] pero este hace referencia al momento en que se realizó una transacción, no se detalla cuándo, en el mundo real, varían los atributos o interrelaciones involucradas en esas transacciones, La necesidad de registrar valores que permitan evaluar tendencias, variaciones, máximos y mínimos, justifican considerar en el diseño del datawarehouse cómo algunos atributos o interrelaciones pueden variar en el tiempo. Un esquema multidimensional temporal que incluya, además del hecho principal de análisis, esquemas temporales (que no pertenezcan a la jerarquía) permitirá registrar, además, la variaciones temporales de atributos y/o interrelaciones.

Para la construcción del esquema temporal [23] se adaptó un algoritmo que permite en forma semiautomática construir, a partir de un modelo entidad interrelación, el diseño conceptual de un datawarehouse [9]. Se utilizó una extensión del modelo entidad interrelación, ampliándolo con atributos e interrelaciones temporales y, aplicando un algoritmo recursivo, se construyó el esquema conceptual, unificando en un sólo modelo, tanto el esquema multidimensional como el temporal; este esquema permite registrar y analizar las variaciones temporales así como la realización de consultas sobre

la estructura multidimensional. La transformación, de un modelo de datos temporal a un modelo multidimensional temporal, se realizó de manera informal en dos etapas: primero, utilizando el algoritmo recursivo, se creó un grafo de atributos; luego, a partir del grafo de atributos más un conjunto de decisiones de diseño para la determinación de cuáles serán dimensiones, jerarquías y atributos de hecho, se derivó el modelo multidimensional temporal. La realización de estos pasos, si bien están detallados en la metodología de diseño propuesta, no están formalizados de manera conveniente para poder ser automatizados.

Model-Driven Architecture [18] fue establecida como una arquitectura para el desarrollo de aplicaciones; tiene como objetivo proporcionar una solución para los cambios de negocio y de tecnología, permitiendo construir aplicaciones independientes de la implementación; representa un nuevo paradigma en donde se utilizan modelos del sistema, a distinto nivel de abstracción, para guiar todo el proceso de desarrollo. La idea clave subyacente es que, si se trabaja con modelos, se obtendrán importantes beneficios tanto en productividad, portabilidad, interoperatividad y mantenimiento. Podemos dividir el proceso MDA en tres fases; en la primera, se construye un modelo independiente de la plataforma (PIM), este es un modelo de alto nivel del sistema, independiente de cualquier tecnología; luego, se transforma el modelo anterior a uno o más modelos específicos de la plataforma (PSM), este modelo es de más bajo nivel que el PIM y describe al sistema de acuerdo con una tecnología de implementación determinada; por último, se genera el código fuente a partir de cada PSM. La división entre PIM y PSM está vinculado al concepto de plataforma que no está, aún, claramente definido. MDA, además, presenta un modelo independiente de los aspectos computacionales (CIM) que describe al sistema dentro de su ambiente y muestra lo que se espera de él sin exhibir detalles de cómo será construido. El beneficio principal del enfoque MDA es que una vez que se ha desarrollado cada PIM podemos derivar, automáticamente, el resto de los modelos aplicando las correspondientes transformaciones en forma vertical. Sin embargo, pueden aplicarse también transformaciones horizontales; esto es, un modelo fuente se puede transformar en un modelo destino dentro del mismo nivel de abstracción [20]. La transformación de PIM a PIM se utiliza cuando los modelos son ampliados o especializados durante el proceso de desarrollo sin necesidad de información dependiente de la plataforma. Una de las más obvias transformaciones es entre el análisis y el diseño, concepto relacionado con el refinamiento de modelos [18]. Aplicando los conceptos de MDA en la construcción de un datawarehouse, identificamos un CIM que define los requerimientos desde una perspectiva de negocio; un PIM que lo define desde un punto de vista conceptual sin tener en cuenta ningún detalle tecnológico específico y uno o más PSM´s que especifican aspectos de diseño en distintas plataformas, por ejemplo, ROLAP (OLAP Relacional), MOLAP (OLAP Multidimensional) u HOLAP (OLAP Hibrido) [17].

En el presente artículo proponemos, dentro del marco de la filosofía MDA, formalizar la transformaciones presentadas; primeramente, una transformación horizontal (de PIM a PIM), del modelo de datos temporal al modelo multidimensional temporal pasando por un grafo de atributos; luego, realizaremos la transformacion vertical (de PIM a PSM), a una plataforma relacional. Utilizaremos un metamodelo para cada uno de los modelos propuestos y aplicaremos sentencias OCL (Object Constraint Language) [25], para formalizar las transformaciones.

El resto del trabajo está estructurado de la siguiente forma: en el capítulo 2, presentamos la transformación informal del modelo de datos al grafo de atributos, del grafo de atributos al modelo multidimensional y de este último al modelo relacional; en el capítulo 3, mostramos los metamodelos de datos, de grafos, multidimensional y relacional y las tres transformaciones formales; en el capítulo 4, detallamos los trabajos relacionados, tanto los referidos al diseño conceptual de un datawarehouse temporal como a las propuestas de diseño de un datawarehouse en un ambiente MDA; por último, en el capitulo 5, presentamos la conclusión y los trabajos futuros.

2. Transformaciones Informales

La metodología de transformación del modelo de datos temporal al modelo relacional plantea una serie de pasos, descritos de manera informal, que detallaremos a continuación y que, en resumen, consisten en la aplicación de un algoritmo que tiene como entrada un modelo entidad interrelación temporal y, mediante sucesivas transformaciones, obtenemos, primeramente, un modelo multidimensional temporal y, finalmente, un conjunto de tablas relacionales. Presentamos con un ejemplo (Figura 1) cómo, utilizando el algoritmo, transformamos un modelo de datos temporal (Figura

2) a un grafo de atributos (Figura 3); luego, a partir de este, creamos el modelo multidimensional temporal (Figura 4) y, finalmente, las tablas en el modelo relacional.

Para la aplicación del algoritmo recursivo, primero, transformamos el modelo entidad interrelación (Figura 1) a un modelo entidad interrelación temporal[1] (Figura 2). El atributo multivaluado se convertirá en una entidad débil con una interrelación temporal (marcada con T) y la interrelación temporal se transformará en una entidad con interrelaciones binarias (marcada con T) vinculadas a las entidades participantes [23]. En los casos en que querramos preservar una futura jerarquía, proponemos mantener las dos interrelaciones (la instantánea y la temporal). En el ejemplo (Figura 2), conservamos la interrelación entre *PROVEEDOR* y *LOCALIDAD*. Con el mismo criterio general utilizado para transformar interrelaciones temporales, transformamos la interrelación *venta* en una entidad *VENTA* (Figura 2).

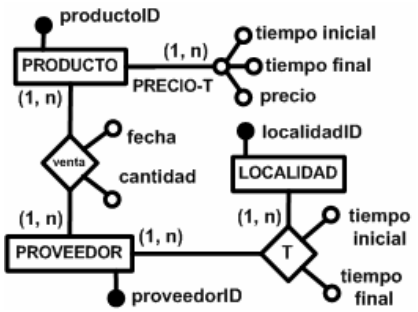


Figura 1. Modelo de Datos

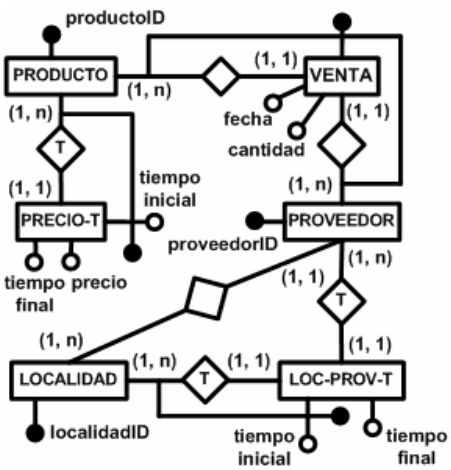


Figura 2. Modelo de Datos Transformado

[1] Por razones de espacio no presentaremos la transformación del modelo de datos al modelo de datos temporal.

2.1 Del Modelo de Datos al Grafo

Los hechos, como conceptos de interés primario para el proceso de la toma de decisión, corresponden a sucesos que ocurren dinámicamente en la realidad, éstos pueden ser representados en el modelo entidad interrelación temporal mediante una entidad E o por medio de una interrelación R n-aria entre entidades E_1... E_n. [9]. Dada un área de interés en un modelo entidad interrelación temporal y una entidad E que pertenece a él, denominamos grafo de atributos al grafo tal que:

- Cada vértice corresponde a un atributo, simple o compuesto del modelo entidad interrelación.
- La raíz corresponde al identificador de E.
- El atributo correspondiente a cada vértice *v*, determina funcionalmente a todos los atributos descendientes de *v*.

Los vértices temporales representan esquemas que tienen como foco de interés la variación de atributos e interrelaciones en función del tiempo. Dado un identifier(E) que indica un conjunto de atributos que identifican a la entidad E, el grafo de atributos (Figura 3) será construido semi automáticamente mediante la aplicación de la siguiente función recursiva modificada de [9]:

```
Function translate (E: Entity): Vertex
{v = newVertex(E);
// newVertex(E) crea un nuevo vértice,
// conteniendo el nombre y el identificador
// del  objeto E
for each attribute a Î E½a Ï identifier(E)
do
   addChild (v, newVertex(a));
   // se agrega un hijo a al vértice v
   for each  entity G connected to E
   by relationship R½card-max(E,R)=1 xor R is
temporal do
// se consideran interrelaciones y atributos
// temporales
{for each  attribute  b Î R do
   addChild (v, newVertex(b));
   addChild (v, translate(G));
}
return(v)}
```

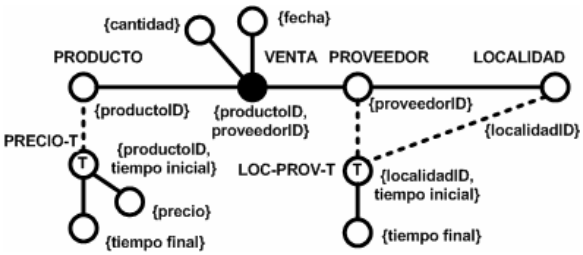


Figura 3. Grafo de Atributos

Cuando ampliamos el modelo entidad interrelación con aspectos temporales, los atributos y las interrelaciones variantes se transformarán en entidades vinculadas con interrelaciones marcadas con T del tipo x-a-muchos (card-max(E, R) > 1); por lo tanto, no podrán ser incluidos en la jerarquía para realizar agregaciones. La línea punteada en el grafo de atributos muestra esta particularidad.

Probablemente no todos los atributos representados en el grafo sean de interés en el datawarehouse. Por tal motivo, este puede ser modificado por el diseñador para eliminar los niveles de detalles innecesarios.

2.2. Del Grafo al Modelo Multidimensional

El proceso de transformación del grafo de atributos al modelo multidimensional temporal, es decir, la elección de cuales vértices del grafo serán atributos de hecho, dimensiones o jerarquías (temporales o no) dependerá de las decisiones del diseñador pero, en general, seguirá el siguiente criterio que utilizaremos en la transformación: la raíz del grafo será el hecho principal; todos los vértices vinculados con la raíz, que no sean identificadores, serán atributos de hecho; los demás atributos vinculados al hecho serán dimensiones; los vértices vinculados a las dimensiones, que no sean identificadores, serán atributos de la dimensión; los demás atributos serán jerarquías (temporales o no) dentro de las dimensiones; todos los atributos vinculados a una jerarquía, si no son identificadores, serán atributos de éstas, sino serán, también, parte de la jerarquía; todas las jerarquías temporales tendrán asociados un rango temporal. El atributo fecha, asociado al hecho, si lo hubiere, será transformado en dimensión. En la figura 4 se muestra el esquema resultante.

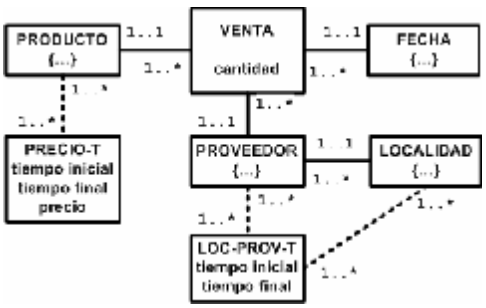


Figura 4. Esquema Multidimensional Temporal

Los atributos e interrelaciones temporales en el grafo (éstos se vinculan mediante líneas punteadas) precisan de una consideración especial en su transformación al esquema de hecho: éstos no formarán parte de la jerarquía para las operaciones de roll-up y drill down, solamente permitirán evaluar cuándo lo atributos e interrelaciones han variado en el tiempo; constituyen, lo que se denomina, jerarquías no estrictas [27].

2.3. Del Modelo Multidimensional al Relacional

Por último, a partir del modelo multidimensional temporal obtendremos, aplicando las siguientes reglas de transformacion, un conjunto de tablas en el modelo relacional. El hecho se transformará en tabla; los atributos de hecho, serán columnas de la tabla; la clave primaria estará compuesta por el conjunto de los atributos identificadores; los atributos que forman la clave primaria serán claves foráneas referenciando a cada una de las tablas resultantes de las transformaciones de las dimensiones del hecho. Las dimensiones se transformán en tablas; los atributos de la dimensiones serán columnas de la tabla; la clave primaria estará compuesta por el conjunto de los atributos identificadores; además, cada tabla dimensión tendrá una clave foránea que hará referencia a cada una de las tablas jerarquía vinculadas a la dimensión. Las jerarquías se transformarán en tablas; los atributos de la jerarquía serán columnas de la tabla; la clave primaria estará compuesta por el conjunto de los atributos identificadores; además, cada tabla jerarquía tendrá una clave foránea que hará referencia a cada una de las jerarquías vinculadas. Las jerarquías temporales se transformarán en tablas. Si la jerarquía temporal deviene de un atributo temporal (isTempAttr = true), tendrá como atributo el tiempo final; la clave primaria será la unión de la clave primaria de la jerarquía vinculada (además, será la clave foránea que hará referencia a dicha tabla jerarquía) más el tiempo inicial. Si la jerarquía temporal deviene de una interrelación temporal (isTempAttr = false), tendrá como atributo el tiempo final y el atributo que es clave primaria de una de las jerarquías vinculada (además, será la clave foránea que hará referencia a la tabla jerarquía); la clave primaria será la unión de la clave primaria de la otra jerarquía vinculada (además, sera la clave foránea que hará referencia a dicha tabla jerarquía) más el tiempo inicial. A continuacion, se presenta el esquema relacional resultante:

VENTA(productoID(PRODUCTO), proveedorID(PROVEEDOR), fechaID(FECHA), cantidad)
FECHA(fechaID,…)
PRODUCTO(productoID, …)

PROVEEDOR(proveedorID, localidadID(LOCALIDAD),…)
LOCALIDAD(localidadID,…)
PRECIO-T(productoID(PRODUCTO), tiempo-inicial, tiempo-final, precio)
LOC-PROV-T(proveedorID(PROVEEDOR), tiempo-inicial, tiempo-final, localidadID(LOCALIDAD)

3. Transformaciones Formales

Una regla de transformación de modelos debe definir, evitando cualquier ambigüedad, la relación implícita que existe entre sus partes. MDA propuso un estándar, QVT (Query, Views, Transformations) [29], que permite crear consultas, vistas y transformaciones de modelos en el marco MDA . Las transformaciones, en el contexto de QVT, se clasifican en *relación* (relation) y *función* (mapping); las relaciones especifican transformaciones multidireccionales, no permiten crear o modificar modelos, pero sí chequear la consistencia entre dos o más modelos relacionados. Las funciones, en cambio, implementan la transformación, es decir, transforma elementos de un dominio en elementos de otro. Se han propuesto varios lenguajes de transformación: BOTL [16]; ATL [14]; Tefkat [15]; Kent Model [1] y también el uso de sentencias OCL para especificar las transformaciones [6], [7]. Todos estos lenguajes asumen que los modelos involucrados en la transformación cuentan con una definición formal de su sintaxis expresada en términos de metamodelos MOF [21]. En este trabajo hemos utilizado el lenguaje declarativo OCL como alternativa a otros lenguajes específicos para expresar la transformación entre los modelos de datos. OCL es suficientemente expresivo y cuenta con una definición más madura y estable que la de los otros lenguajes mencionados.

3.1. Metamodelos

Para la especificación de las reglas de transformación es esencial el conocimiento de los metamodelos, tanto de los modelos fuente como de los modelos destino [18]. UML [30] es ampliamente recomendado y aceptado, aunque no especialmente prescripto, como lenguaje de especificación para modelos MDA.

A continuación, presentaremos los cuatro metamodelos utilizados para las transformaciones: el metamodelo de datos temporal (Figura 5), el metamodelo del grafo de atributos (Figura 6), el metamodelo multidimensional temporal (Figura 7) y el metamodelo relacional (Figura 8). Todas las clases, excepto las del metamodelo del grafo de atributos, heredan el atributo `name` de una superclase `Named`, no mostrada en los graficos.

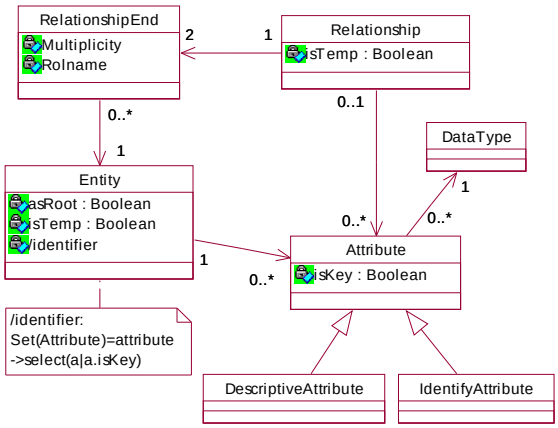


Figura 5. Metamodelo de Datos Temporal

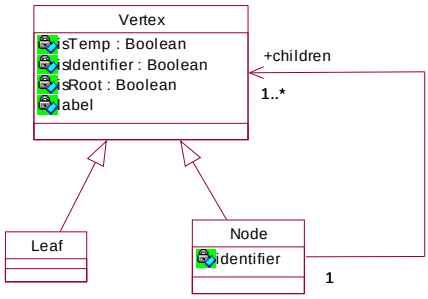


Figura 6. Metamodelo del Grafo de Atributos

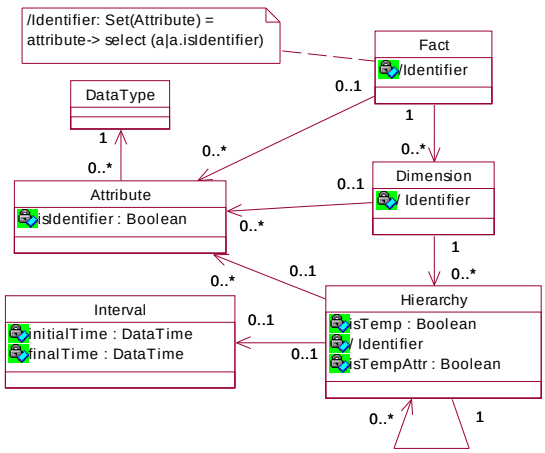


Figura 7. Metamodelo Multidimensional Temporal

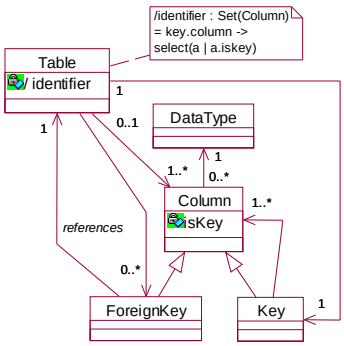


Figura 8. Metamodelo Relacional

3.2. Del Modelo de Datos al Grafo

En esta sección especificaremos formalmente la transformación del modelo de datos al grafo de atributos, la cual fue descrita informalmente en la sección 2.1; utilizaremos los metamodelos de las figuras 5 y 6 y el lenguaje OCL para definir una funcion toVertex() que, al ser aplicada sobre un objeto de tipo Entity perteneciente al modelo de datos temporal, retorna un objeto de tipo Vertex correspondiente al grafo de atributos. La función toVertex() se especifica mediante la siguiente construcción OCL:

```
Context e:Entity :: toVertex(): Vertex
post:  e.name = result.label and
e.identifier = result.identifier
-- los atributos de e se convierten
-- en hijos de result
and e.attribute -> forall(a | e.identifier ->
includes(a) or  result.children ->
includes(toLeaf())
-- Se consideran atributos e
-- interrelaciones temporales
and e.connections -> forAll( Tuple{r, g} |
(card-max(e, r) = 1 xor r.isTemp) implies
(r.attribute -> forall(b | result.children ->
includes(toLeaf()))
and result.children -> includes(g.toVertex())
)
```

Definiciones adicionales al metamodelo:

```
Context Entity :: identifier: Set(Attribute)
-- retorna el conjunto de atributos
-- identificadores de la entidad;

Context Entity :: connections: Set(TupleType
(r: Relationship, g: Entity))
-- retorna el conjunto de todas las posibles
-- tuplas {r, g} donde r es una interrelación
-- vinculada a self, en tanto que g sea una
-- entidad conectada al extremo opuesto de r

Context a: Attribute :: toleaf() : Leaf
result.name = a.name
-- transforma el atributo en hoja del grafo
```

3.3. Del Grafo al Modelo Multidimensional

La transformación del grafo de atributos al modelo multidimensional, tal como fue detallada en la sección 2.2, será formalmente especificada utilizando expresiones OCL sobre los metamodelos de las figuras 6 y 7, de la siguiente forma:

```
Context v: Vertex :: toFact(): Fact
Pre: v.isRoot
-- la raíz del grafo se transforma en  el
-- hecho principal
Post: result.name = v.label
-- los vértices vinculados con la raíz, que
-- no sean identificadores, se transformarán
-- en las  medidas del hecho; los demás
-- atributos vinculados al hecho, serán
-- dimensiones; el atributo fecha, asociado
-- al hecho (si lo hubiere), será
-- transformado en dimension
and v.children -> forAll( w | if
(w.isIdentifier or w.label = "fecha")
then
result.dimension -> includes(w.toDimension())
else
result.attribute -> includes(w.toAttribute())
endif)

Context v: Vertex :: toAttribute(): Attribute
Post: result.name = v.label

Context v: Vertex :: toDimension(): Dimension
Post: result.name = v.label
-- los vértices vinculados a las dimensiones
-- que no sean identificadores,serán tributos
-- de la dimensión; los demás atributos serán
-- jerarquías dentro de las dimensiones
and v.children -> forAll( w | if
w.isIdentifier  then
result.hierarchy -> includes(w.toHierarchy())
else
result.attribute -> includes(w.toAttribute())
endif)

Context v: Vertex :: toHierarchy(): Hierarchy
Post: result.name = v.label
-- las jerarquías temporales tendrán
-- asociados un rango temporal
and v.isTemp implies result.isTemp and
result.interval -> notEmpty()
-- los atributos vinculados a una jerarquía,
-- si no son identificadores, serán atributos
-- de éstas, sino, serán parte  de la
-- jerarquía;
and v.children -> forAll(w | if
w.isIdentifier then
result.hierarchy -> includes(w.toHierarchy())
else
result.attribute -> includes(w.toAttribute())
endif)
```

3.4. Del Modelo Multidimensional al Relacional

La transformación del modelo multidimensional al relacional, tal como fue detallada en la sección 2.3,

será formalmente especificada utilizando expresiones OCL sobre los metamodelos de las figuras 7 y 8, de la siguiente forma:

```
Context f: Fact :: toTable(): Table
Post:
-- el hecho se transforma en tabla
result.name = f.identifier
-- los atributos de hecho serán
-- columnas de la tabla
result.column = f.attribute ->
collect(a | a.toColumn())
-- la clave primaria estará compuesta por el
-- conjunto de los atributos identificadores
result.key = f.attribute ->
select(a | a.isIdentifier) ->
collect (i | i.toColumn())
-- los atributos que forman la clave primaria
-- serán, además, claves foráneas
-- referenciando a cada una de las tablas
-- resultantes de las transformaciones de las
-- dimensiones del hecho.
result.foreignKey = result.key and
result.foreignKey -> collect(k | k.table) =
f.dimension -> collect(d | d.toTable())

Context d:Dimension :: toTable(): Table
Post:
-- Las dimensiones se transformarán en tablas
result.name = d.identifier
-- los atributos de la dimensiones serán las
-- columnas de la tabla
result.column = d.attribute ->
collect(a | a.toColumn())
-- la clave primaria estará compuesta por el
-- conjunto de los atributos identificadores
result.key = d.attribute ->
select(a | a.isIdentifier) ->
collect (i | i.toColumn())
-- cada tabla dimensión tendrá claves
-- foráneas que harán referencia a cada una
-- de las jerarquías vinculadas a la
-- dimensión
result.foreignKey -> collect(k | k.table) =
d.hierarchy -> collect(h | h.toTable())


Context h:Hierarchy :: toTable(): Table
Post:
-- las jerarquías se transforman en tablas
result.name = h.identifier
if h.isTemp = true
then
-- Caso 1: transformacion de jerarquias no
-- temporales: los atributos de la jerarquía
-- serán las columnas de la tabla
result.column = h.attribute ->
collect(a | a.toColumn())
-- la clave primaria estará compuesta por el
-- conjunto de los atributos identificadores
result.key = h.attribute ->
select(a | a.isIdentifier) ->
collect (i | i.toColumn())
-- cada tabla jerarquía tendrá una clave
-- foránea que hará referencia a cada una de
-- las jerarquías vinculadas
result.foreignKey -> collect(k | k.table) =
h.hierarchy -> collect(h | h.toTable())
else
if (isTempAttr = true)
then
-- Caso 2: transformación de jerarquía
-- temporal que proviene de un atributo
-- temporal: tendrá como columna al tiempo
-- final;
result.column = h.attribute ->
union Set{h.interval.finalTime} ->
collect(a | a.toColumn())
-- la clave primaria será la unión de  la
-- clave primaria de la jerarquía más el
-- tiempo inicial
result.key = h.attribute ->
select(a | a.isIdentifier) -> union
Set{h.interval.initialTime}->
collect (i | i.toColumn())
else
-- Caso 3: transformación de jerarquía
-- temporal que deviene de una interrelación
-- temporal: tendrá como columna al tiempo
-- final y al atributo que es clave primaria
-- de la jerarquía vinculada; la clave
-- primaria será la unión de la  clave
-- primaria de la otra jerarquía  vinculada
-- más el tiempo inicial.
endif
endif


context a: Attribute :: toColumn(): Column
result.name = a.name
-- transforma el atributo en columna de la
-- tabla
```

4. Trabajos Relacionados

Se propusieron varias soluciones considerando los aspectos temporales en el Datawarehouse; en [5] se presentó un esquema estrella temporal que difiere del tradicional en cuanto al tratamiento del tiempo, mientras este toma al tiempo como una dimensión más, aquel anula la dimensión tiempo y agrega, como atributos de hecho, el tiempo inicial y el final en cada una de las filas de las tablas del esquema. En [27] se describió, entre las características que un modelo de datawarehouse debería tener, la necesidad de considerar los cambios temporales en los datos y las jerarquías no estrictas. En [19] se presentó el modelo multidimensional temporal y un lenguaje de consulta temporal, donde se agregan marcas de tiempo en las dimensiones o al nivel de instancias (o ambos) para capturar las variaciones en los atributos de las dimensiones.

Entre los trabajos vinculados a la transformación de modelos, en [11] se describió, mediante Meta Object Facility (MOF), la transformación del esquema entidad interrelación al esquema relacional utilizando sentencias OCL para establecer restricciones en el metamodelo. En [4] se plantearon dos fases para la migración de un sistema relacional a un sistema de base de datos orientado a objetos; en la primera, utiliza reglas de transformación para construir un esquema OO que es semánticamente equivalente al esquema

esquema relacional, en la segunda fase ese esquema es usado para generar programas que migren los datos relacionales a una base de datos orientado a objetos. En [10] se estudió la sintaxis y la semántica del modelo entidad interrelación y el modelo de datos relacional y sus transformaciones. En [26] se mostró un marco para representar metadatos acerca de datos fuentes, datos destinos, transformaciones y los procesos y operaciones que crean y administran un datawarehouse. En [2] se estudió el problema de la traducción de esquemas entre diferentes modelos de datos, introducen un formalismo teórico gráfico que permite representar uniformemente esquemas y modelos para comparar diferentes modelos de datos y describir el comportamiento de la traducción. En [23] se estableció una conexión formal entre modelos de datos; se utilizaron técnicas de metamodelo basado en MOF para representar la transformación, mediante un algoritmo, del esquema entidad interrelación temporal al modelo multidimensional temporal; se emplearon diagramas de clases MOF y sus correspondientes reglas OCL para establecer restricciones en el modelo y en el metamodelo. En [28] se definió una estrategia para verificar formalmente la corrección de transformaciones entre modelos en el contexto de MDE.

Existen trabajos donde, específicamente, se utilizó el enfoque MDA para el diseño de un datawarehouse. En [17] se presentó un método estándar e integrado para el diseño de un datawarehouse; se definió el MMD^2A (MultiDimensional Model Driven Architecture) como un enfoque para la aplicación del marco MDA en el modelado multidimensional. En [29] se propuso un método para el diseño conceptual de un datawarehouse, planteado en tres fases: en la primera se extraen un conjunto de esquemas multidimensionales de las bases de datos operacionales mediante reglas de transformaciones definidas en el marco de MDA, la segunda fase está vinculada con la identificación y la elección de los requisitos del usuario; por último, estos requisitos se usan para seleccionar y refinar los esquemas multidimensionales. En [24] se presentó, utilizando metamodelos, reglas de transformacion y aplicando el enfoque MDA, una metodología que convierte un modelo entidad interrelacion temporal en un esquema multidimensioneal temporal.

5. Conclusión y Trabajos Futuros

MDA promueve el uso intensivo de modelos en el proceso de desarrollo, se construyen modelos de los sistemas utilizando primitivas de alto nivel de abstracción, luego estos modelos son transformados hasta obtener código fuente del sistema final. Inicialmente, se crea un modelo independiente de la plataforma (PIM); luego, se transforma el modelo anterior a uno o más modelos específicos de la plataforma (PSM) y, por último, se genera el código a partir de cada PSM. En el presente trabajo se desarrolló una metodología semiautomática para generar un esquema relacional de un datawarehouse temporal (ROLAP) a partir de un modelo de datos temporal; primero se presentó un algoritmo recursivo que permitió armar un grafo de atributos a partir de un modelo de datos; luego, se estableció informalmente la transformación del árbol de atributos al modelo multidimensional y de este al esquema relacional; a continuación, se presentaron los metamodelos del modelo de datos temporales, del grafo de atributos del modelo multidimensional y del relacional. Finalmente, utilizando sentencias OCL, se detallaron las transformaciones formales.

En trabajos futuros se desarrollarán reglas de transformación (mapping) que permitan implementar las relaciones (relation) presentadas en este trabajo y en el desarrollo de plug-ins en la plataforma Eclipse que permita implementar el esquema relacional en diferentes DBMS.

Referencias

[1] Akehurst D.H., Howells W.G.J., McDonald-Maier K.D. Kent Model Transformation Language Proc. Model Transformations in Practice Workshop, part of MoDELS 2005, Montego Bay, Jamaica. 2005

[2] Atzeni P, Torlone R., Schema Translation Between Heterogeneous Data Models in a Lattice Framework. 6th IFIP TC-2 Working Conference on Database Semantics (DS-6), Atlanta, Georgia, 1995

[3] Agrawal R, Gupta A, Sarawagi S., Modeling Multidimensional Databases, Research Report, IBM Almaden Research Center, San Jose, California, 1995.

[4] Behm, A., Geppert, A., Dittrich, K. R. "On the Migration of Relational Schemes and Data Object-Oriented Database System". In Proceedings of Re-Technologies in Information System. Klagenfurt, Austria, Dec 1997.

[5] Bliujute R., Saltenis S., Slivinskas G., and Jensen C. S., Systematic Change Management in Dimensional Data Warehousing. in Proceedings of the Third International Baltic Workshop on Data Bases and Information Systems, Riga, Latvia, 1998.

[6] Cariou, E., Marvie, R., Seinturier, L., & Duchien, L. (2004). OCL for the Specification of Model Transformation Contracts. In J. Bezivin (Eds.), Proceedings of OCL&MDE'2004, OCL and Model Driven Engineering Workshop. Lisbon, Portugal. 2004.

[7] Cariou, E., Marvie, R., Seinturier, L., & Duchien. Model Tranformation Contracts and their Definition in UML and OCL. Technical Report 2004-08, 2004.

[8] Chaudhuri S. and Dayal U., An Overview of Data Warehousing and OLAP Technology, ACM SIGMOD Record 26(1), March 1997.

[9] Golfarelli M., Maio D., Rizzi S., The Dimensional Fact Model: a Conceptual Model for Data Warehouses. International Journal of Cooperative Information Systems, vol 7, n.2&3, 1998.

[10] Gogolla Martin, Lindow Arne, Richters Mark, Ziemann Paul: Metamodel Transformation of Data Models, Workshop in Software Model Engineering, 2002.

[11] Gogolla Martin, Lindow Arne: Transforming Data Models with UML, IOS Press, 2003.

[12] Gupta, H., Harinarayan, V. Rajaraman, A. and J. Ullman. Index Selection for OLAP. Proceeding ICDE 1997.

[13] W. Inmon. Building the Data Warehouse. John Wiley & Sons, 2002.

[14] Jouault, F, Kurtev, I: Transforming Models with ATL. In: Proceedings of the Model Transformations in Practice Workshop at MoDELS 2005, Montego Bay, Jamaica.

[15] Lawley Michael, Steel Jim. Practical Declarative Model Transformation with Tefkat, Lecture Notes in Computer Science, Volume 3844, Jan 2006

[16] Marschall Frank, Braun Meter: Model Transformations for the MDA with BOTL In: Proceedings of the Workshop on Model Driven Architecture: Foundations and Applications, CTIT Technical Report TR-CTIT-03-27, Univeristy of Twente, June 2003.

[17] Mazón Jose Norberto, Trujillo Juan, Serrano Manuel, Piattini Mario: Applying MDA to the Development of Data Warehouses. DOLAP 2005: 57-66.

[18] MDA. Model Driven Architecture. 2004. http://www.omg.org/cgi-bin/doc/formal/03-06-01.

[19] Mendelzon A, Vaisman. A Temporal Query in OLAP. VLDB 2000: 242-253.

[20] Mellor S., Scott K., Uhl A., Weise D. MDA Distilled: Principles of Model-Driven Architecture. Addison-Wesley. 2004.

[21] MOF. Mata Object Facility 1.3. OMG (1999)

[22] Neil Carlos, Ale Juan. A Conceptual Design for Temporal Data Warehouse. 31º JAIIO. Santa Fe. Simposio Argentino de Ingeniería de Software. 2002.

[23] Neil Carlos, Pons Claudia. Formalizing the Model Transformation Using Metamodeling Techniques ASSE Argentinean Symposium on Software Engineering. (33 JAIIO04) September 2004. Cordoba. Argentina.

[24] Neil Carlos, Pons Claudia. Diseño Conceptual de un Datawarehouse Temporal en el Contexto de MDA. XII Congreso Argentino de Ciencias de la Computación. CACIC. San Luis. Argentina. 2006

[25] OCL. Object Constraint Language - version 2.0. 2003.

[26] OMG, ed: The Common Warehouse Metamodel Especification. OMG (2000). www.omg.org.

[27] Pedersen T. B., Jensen C. S, Multidimensional Data Modeling for Complex Data. 1998. ICDE 1999

[28] Pons C. and Garcia D. "An OCL-based Technique for Specifying and Verifying Refinement-oriented Transformations in MDE". Proceedings MoDELS/UML 2006 "Model Driven Engineering Languages and Systems, 9th International Conference, MoDELS 2006, Genoa, Italy, October 2006" LNCS.

[29] OMG. Meta object facility (MOF) 2.0 Query/View/Transformation Specification. OMG Document ptc/05-11-01, Nov 2005.

[30] UML. The Unified Modeling Language Specification – version 2.0. 2003.

[31] Zepeda Leopoldo, Celma Matilde: Aplicando MDA al Diseño Conceptual de Almacenes de Datos. JIISIC 2006: 271-278

Estrategias de Detección de "Feature Envy" en Aplicaciones Java

Carlos Angarita Márquez
Politécnico Grancolombiano
angarita@poligran.edu.co

Silvia Takahashi Rodríguez
Universidad de los Andes
stakahas@uniandes.edu.co

Abstract

Este artículo presenta dos propuestas para detección y corrección automática de malas prácticas de diseño y codificación en aplicaciones Java. Las propuestas de detección planteadas están basadas en el uso de métricas de software obtenidas a partir de un análisis estático del código fuente de la aplicación. La falla de diseño que se pretende detectar es la declaración de métodos en las clases inapropiadas. Se presenta un estudio de validación de las propuestas desarrolladas mediante la confrontación de los juicios sobre fallas de diseño emitidos aplicando la intuición humana en contraparte con los juicios de las estrategias de detección automática. Las propuestas presentadas son el resultado de un trabajo de investigación relacionado con el tema.

Palabras clave:
Prácticas de diseño, Refactoring, Bad Smells, Métricas de Software, Feature Envy, Fallas de diseño, Java, XML.

1. Introducción

Los problemas de mantenibilidad del software involucran desde las fallas en el código fuente hasta la ausencia de documentación actualizada de la estructura del sistema informático. Sin embargo, de todos estos factores, los que tal vez impactan con más fuerza la mantenibilidad del software son las fallas cometidas en la etapa de diseño y las malas prácticas en que suele incurrirse durante la etapa de codificación.

Fowler y Beck publicaron en 1999 su libro "Refactoring: Improving the Design of Existing Code" [5], en el cual presentaron una caracterización de veintidós Malas Prácticas de Diseño y Programación (en adelante MPDP). A estas MPDP le asignaron el nombre de "*Bad Smells*", término cuyo uso se ha generalizado en la literatura informática que abarca el tema. Además de Fowler otros autores han señalado y caracterizado buenas y malas prácticas de diseño que han sido denominadas patrones y anti-patrones de diseño que han supuesto un progreso en el tema de detección de MPDP [4].

La identificación de *Bad Smells* realizada por Fowler fue un avance, por cuanto había definido cuales fallas debían ser buscadas en el código fuente para posteriormente aplicar técnicas que removieran dichas fallas. Sin embargo, existe un problema en la forma como Fowler definió los *Bad Smells*. Estos *Bad Smells* fueron definidos de forma muy general, en términos tales que solo puedan ser identificados con cierta facilidad mediante la aplicación de la intuición humana.

El problema subyace en que no es práctico depender de la intuición humana cuando se pretende revisar aplicaciones que constan de miles de líneas de código.

Es necesario contar con mecanismos que exploren de forma automática el código fuente de las aplicaciones y que a partir de una caracterización formal de los *Bad Smells*, señalen la presencia y ubicación de éstos en el código.

Este artículo resume los resultados obtenidos del trabajo de investigación que realizó Carlos Angarita [1] como tesis de Maestría. El resto de este artículo está organizado así: La sección 3 enumera los objetivos del trabajo. La sección 4 describe la propuesta de solución para los problemas planteados. La sección 5 trata acerca del la estrategia de validación usada. Finalmente, se presentan unas conclusiones.

2. Objetivos

2.1. Objetivo General

El objetivo principal del trabajo de investigación descrito fue diseñar, implementar y evaluar la validez de nuevas estrategias de detección de MPDP en aplicaciones Java a partir de una representación XML del código fuente.

2.2. Objetivos Específicos

Como objetivos específicos se tuvieron los siguientes:

- Diseñar una o varias estrategias para detección de una de las malas prácticas definidas por Fowler (el *Bad Smell* "*Feature Envy*"[1]) que incorporen elementos diferenciadores con respecto a las existentes en la actualidad.
- Desarrollar una herramienta en Java que implemente las estrategias definidas.
- Validar la efectividad de las estrategias de detección de *Bad Smells* propuestas, confrontándolas con la intuición humana.

3. Caracterización del Problema

3.1. Qué es Feature Envy?

Martin Fowler define el *Feature Envy* de la siguiente manera "un método parece estar más interesado en otra clase que en la propia clase a la que pertenece". Partiendo de esta definición, se asume que un proceso de detección automática de *Feature Envy* debería analizar las instrucciones de cada método en las clases de la aplicación evaluada, señalado para cada uno de los métodos si tiene *Feature Envy* y en caso de ser así, sugerir cual clase sería la más apropiada para declararlo. Sin embargo, para hacer una detección automática se exige una definición formal y muy precisa, por lo que se requiere afinar la definición de Fowler. En particular la expresión "estar interesado" debe ser precisada, ya que puede ser entendida de diferentes maneras de acuerdo a la forma como el lector de respuesta a preguntas tales como las formuladas a continuación

- ¿Estar interesado en otra clase X corresponde a realizar en tiempo de ejecución un volumen alto de invocaciones a atributos o métodos de dicha clase X?
- ¿Estar interesado en otra clase X corresponde a tener muchos objetos instanciados a partir de dicha clase X?
- ¿Estar interesado en otra clase X corresponde a tener accesos directos a muchos atributos de dicha clase X?
- ¿Estar interesado en otra clase X corresponde a tener accesos a muchos métodos de dicha clase X?
- ¿Se considera que hay un acceso a la clase X, cuando se utiliza un atributo de esta clase X el cual es instancia de una clase Y, para a través de dicho atributo llegar a un atributo de la clase Y?
- ¿Se considera que hay un acceso a la clase X, cuando se utiliza un método de X que retorna un objeto instancia de una clase Y, para a través de dicho objeto acceder a otro método de la clase Y?
- ¿En casos como los dos anteriores se considera que hay accesos a ambas clase X e Y, o solamente se considera que hay acceso a la clase propietaria del atributo cuyo valor es modificado o consultado?
- Si se tiene que la clase X hereda de la clase X1, y a través de un objeto instancia de la clase X, se accede a un atributo o método declarado en la clase X1, se considera que se accedió a la clase X, o a X1 o a ambas?
- ¿El interés en otra clase X se mide por la cantidad de atributos accedidos de ésta o por el número de invocaciones a dichos atributos?
- ¿El interés en otra clase X, se mide por la cantidad de métodos accedidos de ésta o por el número de invocaciones a dichos métodos?
- ¿Cuando se va a medir el "interés" en otra clase, se valoran por igual los accesos a atributos y los accesos a los métodos de ésta?
- ¿Los accesos a los métodos del tipo get o set son tratados como accesos a métodos o a atributos?
- ¿Es válido valorar el "interés" en otra clase, según el porcentaje de las funcionalidades que esa otra clase este poniendo al servicio de la clase solicitante?
- ¿Es válido valorar el "interés" en otra clase, según el porcentaje de instrucciones del método que están apoyadas en la otra clase?

El aporte fundamental de este trabajo, consiste en el análisis a profundidad del significado del *Feature Envy* mediante el planteamiento de una estrategia de detección que considere aspectos como los puestos de manifiesto en las preguntas anteriores. En realidad, más que una propuesta rígida de detección, lo que se ha planteado es un modelo de detección parametrizable según las respuestas que el lector brinde a los cuestionamientos mencionados.

Frente al *Feature Envy* se plantearon dos estrategias de detección parametrizables que se explicarán con todo detalle a continuación.

4. Propuesta de Solución

En este artículo se presentan dos estrategias para lograr la de detección automática de *Feature Envy*, denominadas "Nivel de Explotación de Clases" y "Porcentaje de Invocaciones". Ambas estrategias se basan en el análisis de las instrucciones contenidas en cada uno de los métodos de la aplicación java que se esta evaluando. El mencionado análisis del código fuente no es realizado sobre la representación nativa del código en Java, sino en una representación XML del mismo, la cual se obtiene mediante un proceso de transformación del código fuente original usando la herramienta "XSCoRe" [6]. En la Figura 1 se puede observar el esquema de operación de la detección automática en las dos estrategias planteadas.

1 En la sección 3.1 se define el "*Feature Envy*".

En ambas estrategias el proceso de detección se basa en la obtención inicial de unas métricas referidas al código fuente, de manera similar a como se ha realizado en trabajos previos de otros investigadores que han abordado el tema [3][7][8]. Sin embargo, hay dos características muy particulares en este enfoque como son:

- Utilización de una representación xml del código fuente.
- Utilización de métricas que no pueden obtenerse por introspección del código fuente y que exigen un análisis profundo y detallado de las instrucciones del cuerpo de los métodos de la aplicación que esta siendo evaluada.

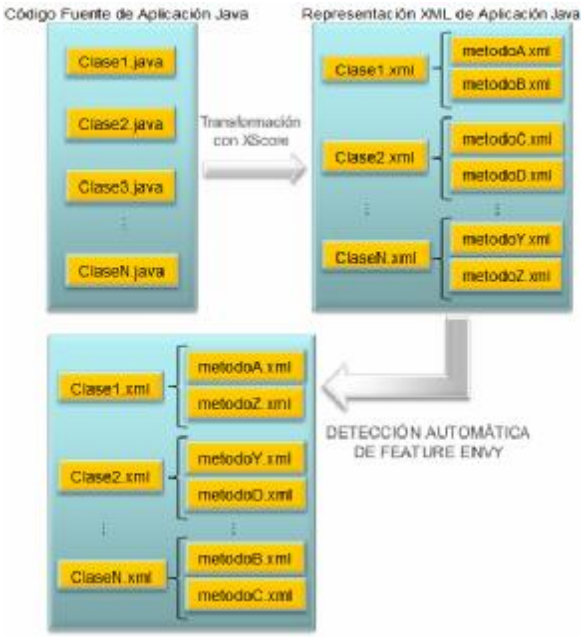


Figura 1. Esquema de Operación

4.1. Estrategia 1: Nivel de explotación de Clases

El *Feature Envy* cuestiona la presencia de un determinado método en una clase, sugiriendo la posibilidad de transportarlo a otra.

Se dice que un método incorpora *Feature Envy* cuando éste requiere el uso de gran cantidad de información correspondiente a atributos o métodos de objetos de otras clases y relativamente poca información tomada de los propios atributos o métodos de la clase a la cual es declarado el método que se esta analizando.

La estrategia de detección propuesta permite lograr simultáneamente dos cosas:

- Identificar si el método incorpora el *Feature Envy*.
- Identificar a cuál clase debería ser transportado el método, bajo criterios de nivel de explotación de clase.

4.1.1. Mecanismo de detección.

Suponga que se tiene una clase X, la cual declara un método m1() al cual se le realizará el análisis de *Feature Envy*. Dentro del conjunto de instrucciones del método m1() participan gran cantidad de objetos, los cuales son instanciados a partir de un conjunto de clases. A este conjunto de clases se le llamará "Set1". Dentro del conjunto "Set1", también se halla incluida la clase propietaria del método m1(). Cada una de las clases contenidas en el conjunto "Set1", tiene naturalmente un conjunto de atributos y métodos, de los cuales solo una porción son accesibles desde el método m1(). Esta porción representa el 100% de la funcionalidad que puede exportar dicha clase al método m1().

La estrategia de detección se basa en que para cada una de las clases del conjunto "Set1", se calculen dos métricas. La primera métrica indica qué porcentaje de los atributos en dicha clase (que cumplan la condición de ser accesibles desde el método m1()) se están utilizando en las instrucciones del método m1(). A esta métrica se le llamará "Factor de utilización de Atributos" (en adelante FUA). La segunda métrica es análoga a la anterior, pero determinando el porcentaje de métodos accesibles que son utilizados desde el método m1(). A esta segunda métrica se le llamará "Factor de utilización de Métodos" (en adelante FUM).

A partir de las métricas FUA y FUM mencionadas anteriormente se puede obtener una tercera métrica que se ha denominado "Factor de utilización de Clase" (en adelante FUC). Dado que FUA y FUM son métricas cuyos valores están contenidos en el rango de 0 a 1, y que se desea que la tercera métrica FUC, también este en el rango de 0 a 1, se utiliza una fórmula clásica de asignación de pesos a FUA y a FUM para calcular el FUC. La fórmula para calcular el FUC es la siguiente:

FUC = (PesoAtributos * FUA) + (PesoMetodos*FUM)
Donde:
(i) 0 <=PesoAtributos,PesoMetodos<=1
(ii) PesoAtributos+PesoMetodos = 1

El valor del FUC indica qué porcentaje de la funcionalidad que ofrece una clase, esta siendo utilizado por el método m1().

La idea en la que se basa esta estrategia de detección es que del conjunto de clases "Set1", la clase que posea un FUC más alto es la que debería declarar el método m1(), por cuanto es la clase que mayor porcentaje de sus funcionalidades esta aportando en el

cumplimiento de las tareas realizadas por el método m1().

4.1.2. Métricas utilizadas

Para efectuar el proceso de detección, primero se debe recopilar un grupo de métricas referentes a cada clase del código fuente de la aplicación como se observa en la Tabla 1. A continuación se describe el significado de cada una de las métricas señaladas en dicha tabla.

Tabla 1. Métricas en detección de "***Feature Envy***"

NC	NAA	NMA	NAU	NMU	FUA	FUM	FUC
ClassX	4	8	0	6	0/4	6/8	0.5(0/4)+0.5(6/8)
ClassY	5	10	4	4	4/5	4/10	0.5(4/5)+0.5(4/10)
ClassZ	7	11	5	3	5/7	3/11	0.5(5/7)+0.5(3/11)

NC (Nombre de Clase)

Esta columna relaciona los nombres de las diferentes clases a partir de las cuales se instanciaron los objetos utilizados en las instrucciones del método que se esta analizando. Estos nombres corresponden a las clases del proyecto, es decir no se incluyen clases pertenecientes a archivos Jar externos, ni clases propias de librerías de Java. A estas clases se les llamará en adelante clases miembro por "invocación directa". Los datos consignados en la columna NAA y sucesivas, están referidos a la clase que figura en la columna NC. Las métricas referentes a una misma clase, se consignan en una misma fila.

Adicionalmente a las clases incluidas por invocación directa, la tabla de métricas puede incluir otros nombres de clases a las que se les llamará en adelante clases miembro por "invocación indirecta", si llega a presentarse alguna de las situaciones que se mencionan a continuación:

Caso 1. Cuando en el método analizado se tiene una instrucción como:

```
this.nameAttribute1.nameMethod1();
```

Si el atributo nameAttribute1 corresponde a un objeto instancia de una clase Z del proyecto, el nombre de dicha clase se incluye en el listado de clases a las cuales se le calcularán métricas. La clase Z es incluida como una clase miembro por *invocación indirecta.*

Caso 2. Cuando en el método analizado se tiene una instrucción como:

```
obj1.nameMethodX().nameMethodY().nameMethodZ();
```

Supóngase que el objeto obj1, es instancia de una clase X, la cual tiene un método llamado nameMethodX. Este método retorna un objeto instancia de una clase que se llamará Y, la cual tiene un método llamado nameMethodY. Este método a su vez retorna un objeto instancia de una clase que se llamará Z, la cual tiene un método llamado nameMethodZ.

Este ejemplo que se ha planteado con tres clases X, Y y Z, podría extenderse con muchas más; lo importante a resaltar aquí es que de éste grupo de clases, se toman las que pertenezcan al proyecto y se anexan a la tabla de clases a las cuales se les calcularán métricas. Las clases X, Y y Z son incluidas como clases miembro por *invocación indirecta.*

Caso 3. Cuando se tiene en el método analizado una instrucción como:

```
obj1.nameAttribute1=5;
```

Supóngase que el objeto obj1, es instancia de una clase X, pero el atributo nameAttribute1, no es declarado en la clase X, sino en una clase ancestro de X, que se llamará W. En este caso la clase W, se incluye en el listado de clases a las cuales se le calcularán métricas. La clase W es incluida como clase miembro por *invocación indirecta.*

NAA (Número de Atributos Accesibles)

Esta columna relaciona el número de atributos de la clase que son accesibles desde el método al cual se le esta haciendo el análisis de *Feature Envy*. El procedimiento para determinar cuáles son dichos atributos se describe a continuación.

Supóngase que se llama X la clase que declara el método sobre el cual se esta realizando el análisis de *Feature Envy*, y Y el nombre de la clase que declara los atributos a los que se les esta realizando un análisis para saber si son accesibles desde la clase X. Se asume que un atributo de la clase Y es accesible desde la clase X si cumple alguna de las siguientes condiciones:

- Si las clase X e Y son la misma.
- Si el atributo tiene modificador de acceso "public".
- Si el atributo tiene modificador de acceso "protected" o sin especificar y las clase X e Y pertenecen al mismo paquete.
- Si el atributo tiene modificador de acceso "protected" y la clase X hereda de la clase Y.

NMA (Número de Métodos Accesibles)

Esta columna relaciona el número de métodos de la clase que son accesibles desde el método al cual se le esta haciendo el análisis de *Feature Envy*. En este conteo de métodos no se contabilizan los constructores, ni los métodos del tipo Get o Set. El procedimiento para determinar cuáles son los métodos accesibles es el análogo al descrito en la sección anterior para identificar los atributos accesibles.

El procedimiento para determinar si un método es del tipo *Get* o *Set*, consiste en la comparación de las instrucciones del mismo con una plantilla de la estructura inicial del método.

Características de identificación de Patrón "Get"

- Se ignora el nombre del método y los parámetros que recibe. Esto significa que no se aplica la técnica tradicional de buscar un nombre de método compuesto por el prefijo "Get", seguido del nombre de un atributo.
- Se ignoran las líneas en blanco y los comentarios de una o varias líneas.
- Se exige que la primera línea de código del método tenga una estructura como la que se muestra a continuación.

```
return attributeName;
```

- Se ignora la existencia de múltiples espacios como separadores en la expresión, al igual que la posible ubicación del "`return`" y el "`attributeName`" en líneas diferentes.

Características de identificación de Patrón "Set"

- Se ignora el nombre del método y los parámetros que recibe. Esto significa que no se aplica la técnica tradicional de buscar un nombre de método compuesto por el prefijo "Set", seguido del nombre de un atributo.
- Se ignoran las líneas en blanco y los comentarios de una o varias líneas.
- Se exige que la primera línea de código del método tenga una estructura como se muestra a continuación.

```
attributeName=[expression];
```

- Se ignora la existencia de múltiples espacios como separadores en la expresión, al igual que la posible ubicación del "attributeName=" y la expresión del lado derecho de la asignación en líneas diferentes.

NAU (Número de Atributos Utilizados)

Esta columna relaciona el número de atributos utilizados en las instrucciones del método que se esta analizando. Es importante aclarar que aquí se contabilizan los accesos ya sean directos o indirectos. A continuación se muestra un ejemplo para clarificar lo que es llamado acceso indirecto a un atributo.

```
[Instrucción 1]
Obj1.nameAttribute1InClassX.nameAttribute1InClassY=5
[Instrucción 2]
Obj1.nameMethod1InClassX().nameAttribute1InClassY=5
```

En la Instrucción 1, se tiene un objeto llamado `Obj1` que es instancia de una clase que se llamará X. Dicha clase tiene un atributo llamado `nameAttribute1InClassX` que es un objeto instancia de una clase que se llamará Y, la cual a su vez tiene un atributo numérico llamado `nameAttribute1InClassY`. En esta instrucción se contabiliza un acceso a un atributo en la clase X considerado acceso directo y un acceso a un atributo de la clase Y considerado acceso indirecto.

En la Instrucción 2, se tiene un objeto llamado `Obj1` que es instancia de una clase que se llamará X. Dicha clase tiene un método llamado `nameMethod1InClassX()` que retorna un objeto instancia de una clase que se llamará Y, la cual a su vez tiene un atributo llamado `nameAttribute1InClassY`. En esta instrucción se contabiliza un acceso a un método en la clase X considerado acceso directo y un acceso a un atributo de la clase Y considerado acceso indirecto.

También son considerados accesos indirectos, los efectuados a atributos o métodos de clases ancestras.

```
[Instrucción 3]
Obj1ClassX.nameAttribute1InClassW=5;
```

En la instrucción 3, se tiene un objeto llamado Obj1 que es instancia de una clase que se llamará X. Dicha clase tiene como ancestra una clase llamada W, la cual tiene un atributo llamado `nameAttribute1InClassW`. En esta instrucción se contabiliza únicamente un acceso a un atributo de la clase W considerado acceso indirecto.

NMU (Número de Métodos Utilizados)

Esta columna relaciona el número de métodos utilizados en las instrucciones del método que se esta analizando. Es importante aclarar que aquí se contabilizan los accesos ya sean directos o indirectos. En el numeral anterior se presenta un ejemplo claro de dichos conceptos.

FUM (Factor de Utilización de Métodos)

Esta métrica no se calcula a partir del análisis del código fuente sino como el cociente de NMU entre NMA. En caso que NMA tenga un valor de cero, se le asigna un valor de cero a FUM.

Lo que cabe resaltar para esta métrica es su significado conceptual, el cual corresponde al porcentaje de los métodos funcionales[2] que se están utilizando de la clase a la cual se le calcula la métrica.

FUA (Factor de Utilización de Atributos)

Esta métrica es el cociente entre NAU y NAA. En caso que NAA tenga un valor de cero, se le asigna un valor de cero a FUA.

Conceptualmente esta métrica corresponde al porcentaje de los atributos accesibles que se están utilizando directamente o a través de métodos del tipo

[2] En este artículo se denominan métodos funcionales de una clase a todos aquellos métodos accesibles que no son constructores ni *Getters* ni *Setters*.

Getter o Setter en la clase a la cual se le calcula la métrica.

FUC (Factor de Utilización de Clase)

Esta métrica se calcula a partir del FUA y el FUM como se explicó en la sección "Mecanismo de detección". Como se indicó anteriormente ésta es la métrica que finalmente se usa para determinar la presencia o ausencia del *Feature Envy* y también la que indica como aplicar el Refactoring correspondiente.

El FUC se puede interpretar como el nivel de pertenencia de un método a una clase, en función de la dependencia que tiene dicho método de los atributos y métodos de dicha clase.

Es bien conocido que el *Feature Envy* se corrige trasladando el método afectado a la clase más apropiada, y esta estrategia de detección sugiere que la clase más apropiada es aquella que presente un FUC más alto.

En las secciones anteriores se ha descrito la forma como se calculan normalmente las métricas involucradas en la detección del *Feature Envy*. Si bien este es el modo de operación normal que se recomienda, la herramienta construida para implementar dicha estrategia de detección puede ser parametrizada de manera que funcione un poco diferente.

4.2. Estrategia 2: Porcentaje de Invocaciones

Se dice que un método incorpora *Feature Envy* cuando éste se caracteriza porque en la mayoría de sus instrucciones se accede más a otra clase (a través de sus atributos y métodos) que a la clase propietaria del método que se está analizando.

La estrategia de detección propuesta permite lograr simultáneamente dos cosas:

- Identificar si el método incorpora *Feature Envy*.
- Identificar a cual clase debería ser transportado el método, bajo criterios de número de invocaciones a la clase.

4.2.1. Mecanismo de detección

Esta estrategia de detección se basa en un principio muy simple. Se contabilizan el número total de accesos que se hacen a diferentes clases, ya sea de forma directa o indirecta, y luego simplemente se estima qué porcentaje de ellos corresponden a cada clase y la que arroje un mayor porcentaje es considerada como la clase que debería alojar al método que se esta analizando.

En esta estrategia se aplican los mismos principios señalados en la estrategia anterior:

- Se contabilizan los accesos *directos* o *indirectos* a atributos o métodos.
- No hay discriminación en los accesos a las clases. Los accesos a atributos o métodos, ya sean del tipo getters, setters o funcionales, todos se valoran por igual. Todos cuentan como accesos a la clase.

4.2.2. Métricas utilizadas

TAC (Total de Accesos a la Clase)

El TAC es la sumatoria de los accesos a los atributos y a los métodos de cualquier tipo (Funcionales, Getters o Setters).

TA (Total de Accesos)

Esta métrica se define a nivel del método en el que se esta analizando la presencia del *Feature Envy*.

El TA es la sumatoria de los TAC de todas las clases que son invocadas directa o indirectamente desde las instrucciones del método.

EL PU se calcula para cada clase, y es el cociente entre el TAC de la clase y el TA del método en el que se esta analizando la presencia del *Feature Envy*.

Si existe una clase que tenga un valor de PU superior al de la clase la clase que declara el método analizado, se dice que dicho método incorpora el *Feature Envy*.

PU (Porcentaje Utilización)

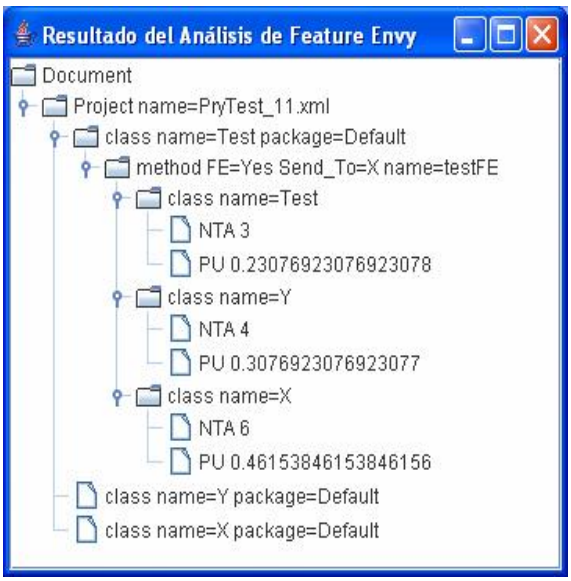


Figura 2. Resultados estrategia 2 Feature Envy

En la Figura 2 se muestra una pantalla típica del resultado de la evaluación de *Feature Envy* mediante esta estrategia de detección. Los resultados allí presentados corresponden al análisis de la aplicación que se muestra en la Tabla 2.

5. Validación de las propuestas

Con el fin de validar la efectividad de las propuestas de detección, se efectuó una comparación entre los conceptos emitidos por parte de un grupo de programadores experimentados frente a los conceptos arrojados por la herramienta que implementó las estrategias de detección propuestas. Los conceptos emitidos se basaron en el análisis del código fuente de un conjunto de catorce casos cuidadosamente seleccionados, cuya presentación es similar a indicada en la Tabla 2.

Los catorce casos evaluados, fueron estratégicamente preparados de manera que cumplieran con las siguientes características:

- Ser muy cortos en términos del número de clases y líneas de código de cada clase.
- Utilizar una notación de fácil comprensión y uniforme en todos los casos.
- Ir acompañados de un diagrama de clases UML que facilite su comprensión.
- Que cada uno apunte a indagar en el encuestado la respuesta a una pregunta específica de las citadas en la sección "*Qué es Feature Envy*".
- Incluir casos de evaluación de control para detectar eventuales diligenciamientos irresponsables de la encuesta o casos de desconocimiento total del tema.

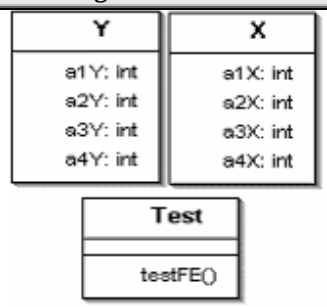
La encuesta señalada se aplicó a través de Internet y fue redactada tanto en idioma español como en inglés para cubrir una población más amplia.

Como mecanismo para dar a conocer la encuesta entre la comunidad informática de desarrolladores se utilizaron los siguientes mecanismos.

- A través de las listas de distribución de correo de la Asociación Colombiana de Ingenieros de Sistemas, ACIS.
- Mediante las listas de distribución de correos de docentes en las principales universidades Colombianas que tiene facultad de ingeniería de sistemas.
- Mediante listas de correo de grupos de programadores Java en diferentes tópicos.
- Mediante listas de correo de desarrolladores interesados en el tema de *Refactoring* y Programación Extrema.

Tabla 2. Caso de prueba

Código Fuente	Diagrama de Clases

`public class Test{`
`int t1,t2,t3;`
`public void testFE(){`
`X obj1X=new X(); X obj2X=new X(); X obj3X=new X();`
`X obj4X=new X(); X obj5X=new X(); X obj6X=new X();`
`Y objY=new Y();`
`t1 = obj1X.a1X + obj2X.a1X + obj3X.a1X;`
`t2 = obj4X.a1X + obj5X.a1X + obj6X.a1X;`
`t3 = objY.a1Y + objY.a2Y + objY.a3Y + objY.a4Y;`
`}`
`}`
	

5.1. Aislamiento de Ruido

Cuando se diseñó la encuesta se contempló el riesgo de que los resultados de ésta se vieran afectados por situaciones como los siguientes:

- Diligenciamientos deliberadamente irresponsables de la misma.
- Diligenciamientos por parte de personas que no tienen conocimiento de programación Orientada a Objetos, o que no comprendan el concepto central de *Feature Envy*.
- Diligenciamientos incompletos de la encuesta.
- Diligenciamientos ignorando las instrucciones que se brindan para dicho proceso.

Para evitar que situaciones como las mencionadas anteriormente le incorporen ruido a la encuesta, se aplicaron los mecanismos de validación.

Los casos uno y cuatro son casos en los que resulta muy evidente que no existe *Feature Envy*. Estos dos casos han sido incluidos únicamente con el propósito de ser herramientas de control. Por lo anterior se optó por descartar las encuestas en las que el desarrollador haya respondido que si hay *Feature Envy* en uno o ambos casos. Estos dos señuelos corresponden a los casos iniciales por cuanto pueden ser parte de encuestas que hayan sido diligenciadas solo parcialmente.

5.2. Resultados Obtenidos

En la Tabla 3, se resumen los resultados obtenidos tras la aplicación de la encuesta Web [2]. De los resultados allí presentados, cabe resaltar que aproximadamente el 50% de las encuestas diligenciadas fueron descartadas por cuanto en éstas, los casos uno y cuatro, es decir los casos usados como herramienta de control evidenciaron que los encuestados o no tenían una cabal comprensión del tema consultado o estaban realizando un diligenciamiento irresponsable de la misma.

Tabla 3. Estadísticas de encuesta web

Concepto	Métrica
Fecha de apertura de la encuesta	Jul 1/05
Fecha de cierre de la encuesta	Ago 31/05
Valor promedio de años de experiencia de quienes diligenciaron encuestas válidas	5.6
Número de personas que usan Programación Extrema	45
Número de personas que No usan Programación Extrema	178
Número de casos válidos tomados para generar estadísticas (cada encuesta tiene 14 casos)	1605
Número de casos eliminados por tener respuesta incorrecta en los señuelos (cada encuesta tiene 14 casos)	1412
% de coincidencia en Estrategia 1	90.5%
% de coincidencia en Estrategia 2	93.1%

6. Conclusiones

En este artículo se presentan los resultados de un trabajo de investigación que trató el problema de detección automática de fallas de diseño. Estas fallas o *Bad Smells* como se conocen normalmente hacen difícil el mantenimiento. Como prueba de concepto nos centramos en un *Bad Smell* en particular: el *Feature Envy*. Se propusieron e implementaron dos estrategias de detección automática, cuya validez se corroboró comparando los resultados de la herramienta de detección con la intuición humana.

A pesar que la estrategia de detección 1, es notablemente más compleja y elaborada que la estrategia de detección 2, esta última arrojó un porcentaje de coincidencia con la intuición humana, ligeramente mayor al obtenido con la estrategia 1.

Quedó comprobada la factibilidad de construir herramientas que detecten de manera rápida y automática malas prácticas de diseño o programación en aplicativos de cientos de miles de líneas de código a partir de la utilización de métricas de software, basadas en análisis estático del código.

Las actividades que normalmente están basadas en la aplicación de la intuición humana para análisis de calidad en diseño y codificación, son factibles de ser representadas mediante modelos matemáticos que brinden un porcentaje muy alto de precisión en los juicios de evaluación del código.

7. Referencias

[1] Carlos Eduardo Angarita Márquez, Tesis de Maestría: "Detección de *Bad Smells* en aplicaciones java a partir de una representación XML del código fuente." Bogotá, Universidad de los Andes, 2005.

[2] Carlos Eduardo Angarita Márquez, "Detección de fallas de diseño en aplicaciones Java." Bogotá, Revista SISTEMAS, No 93 Julio-Septiembre de 2005. ISSN 0120-5919

[3] Beatriz Eugenia Florián Gaviria, Tesis de Maestría: "Detección de *Bad Smells* en Java utilizando métricas de software. " Bogotá, Universidad de los Andes, 2005.

[4] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994.

[5] Martin Fowler. Refactoring: Improving the Design of Existing Code. Addison-Wesley, 1999

[6] Pablo Montes, Tesis de Maestría: "XSCoRe, Representación de Código Fuente basada en XML como una Herramienta de Asistencia al Proceso de Ingeniería en Reversa", Universidad de los Andes, 2004.

[7] Radu Marinescu, "Detection Strategies: Metrics-Based Rules for Detecting Design Flaws," Proceedings of the 20th IEEE International Conference on Software Maintenance (ICSM 2004), IEEE Computer Society Press, ISBN 0-7695-2213-0, pág. 350 -359, 2004

[8] Tom Mens, Tom Tourwe. A survey on software refactoring, Transactions on Software Engineering, IEEE Computer Society Press, February 2004.

8. Autores

Carlos Eduardo Angarita Ingeniero de Sistemas de la Universidad Industrial de Santander; Especialista en Tecnologías Avanzadas para Desarrollo de Software de la Universidad Autónoma de Bucaramanga; magíster en Ing. de Sistemas y Computación, Universidad de los Andes, Bogotá. Actualmente se desempeña como docente investigador en algunos tópicos de la ingeniería de software como el aseguramiento de calidad y reingeniería de software.

Silvia Takahashi Ingeniero de Sistemas y Computación de la Universidad de los andes, Bogotá, Colombia, MSc, Ph.D. de Tulane University, Lousiana, USA. Actualmente es Profesor Asociado del Departamento de Ingeniería de Sistemas y Computación de la Universidad de Los Andes, Bogotá, Colombia. Pertenece al grupo de investigación TiCSw ().

Un caso práctico en MDA para construir aplicaciones JEE5 y .NET

Andrés Yie, Juan Bohórquez, Rubby Casallas
Universidad de los Andes
Grupo de Construcción de Software
a-yie, eu-bohor, rcasalla@uniandes.edu.co

Resumen

En este artículo presentamos un caso práctico de utilización del enfoque Model Driven Architecture (MDA) para construir aplicaciones JEE5 y .NET. El contexto de nuestro trabajo es la construcción de aplicaciones empresariales distribuidas, restringidas a una simplificación de la arquitectura MIS (Management Information Systems).

En nuestra solución utilizamos MDA para desarrollar, sobre JEE5 y .Net, un sistema de administración de configuraciones, que se enmarca dentro de las aplicaciones definidas anteriormente. Para esto utilizamos una estrategia de refinamiento en cuatro dominios para ir componiendo el modelo de nuestra aplicación con las decisiones de diseño y los conceptos de negocio, arquitectura, plataforma y lenguaje de forma incremental. Esta composición se hace por medio de transformaciones automáticas entre modelos, y se finaliza con una generación de código Java, VisualBasic o C# según la plataforma escogida.

1. Introducción

El desarrollo de software empresarial es cada vez más exigente y competitivo, obligando a las empresas desarrolladoras de software a buscar alternativas que les permitan minimizar el consumo de recursos, tener procesos de desarrollo más ágiles y eficientes, y mantener altos niveles de calidad. Por otra parte, la continua y rápida evolución tecnológica a la que se ven enfrentadas las empresas, hace que sea necesario en muchas ocasiones reemplazar completamente las aplicaciones actuales y volver a entrenar o cambiar a los desarrolladores para poder acceder a nuevas funcionalidades tecnológicas.

Con el objetivo de optimizar el desarrollo de aplicaciones y de minimizar los costos de los cambios tecnológicos a los que estamos expuestos en la actualidad, surge una técnica de desarrollo de software basada en modelos como elementos estructuradores de las aplicaciones. Esta propuesta es conocida como MDE (Model Driven Engineering) [1]. Al tomar fuerza esta iniciativa, varias empresas, organizaciones e investigadores empezaron a proponer estándares y especificaciones para la aplicación de los conceptos de modelaje en el mundo real. Entre estas propuestas podemos encontrar a los DSL (Domain Specific Languages) desarrollada por Microsoft [2], al igual que la de OMG (Object Management Group), llamada MDA (Model Driven Architecture) [3].

Una de las soluciones existentes para el desarrollo orientado a modelos es MDA (Model Driven Architecture) [4], la cual es presentada por la OMG (Object Management Group). MDA utiliza los modelos como elementos de primera clase en el desarrollo de aplicaciones, a diferencia de otros paradigmas que los utilizan únicamente como elementos de representación, comunicación y documentación.

Adicionalmente, MDA plantea separar los conceptos del negocio y de la plataforma en dos modelos diferentes: el PIM (Platform Independent Model), y el PSM (Platform Specific Model), y por medio de una transformación convertir el modelo del negocio, en un modelo del mundo en el dominio de la plataforma en la que se va a implementar [5]. Esta solución logra un gran avance usando la abstracción de conceptos, la separación de dominios y las transformaciones automáticas como estrategia, dándonos una ruta para la correcta implementación del desarrollo orientado a modelos en el mundo real.

A pesar de los muchos trabajos recientes en MDA, específicamente en la definición de lenguajes de transformación y otras herramientas [6-8], la propuesta OMG es poco precisa en su implementación, proceso y herramientas. Desafortunadamente, no existen muchos casos

prácticos, completos y documentados de su aplicación.

En este artículo presentamos un caso práctico y completo de la implementación de MDA, intentando llenar el vacio presentado anteriormente. Este es el desarrollo de un sistema de administración de configuraciones que se enmarca dentro de un subconjunto de aplicaciones MIS (Management Information Systems). Este desarrollo se lleva a cabo por medio de la definición de cuatro dominios (negocio, arquitectura, plataforma y negocio), en los cuales se modela el sistema de información deseado, con los cuales se van componiendo los diferentes conceptos que integran una aplicación en plataformas JEE5 y .NET. Esta composición parte de la creación de un modelo del sistema utilizando los conceptos del negocio y luego, por medio de transformaciones automáticas ir componiendo los conceptos de arquitectura, de plataforma y de lenguaje, para finalmente generar código Java, C# o VisualBasic .Net.

Adicionalmente, este artículo presenta los diferentes retos a los que nos enfrentamos buscando generalizar con éxito el desarrollo MDA.

Este artículo está estructurado de la siguiente forma: La sección 2 profundiza sobre MDA y sus retos, buscando enmarcar los problemas a los cuales se enfrenta un desarrollo orientado a modelos. La sección 3 presenta el caso de estudio, y los dominios que se utilizaron para modelar la aplicación deseada. La sección 4 presenta las transformaciones entre los dominios y la generación del código necesarios para la implementación de la aplicación deseada. La sección 5 presenta las herramientas utilizadas en este proceso y la sección 6 las lecciones aprendidas durante esta investigación.

2. Los Retos de MDA

MDA define un conjunto de estándares no propietarios que especifican tecnologías interoperables con las cuales realizarán desarrollos orientados a modelos con transformaciones automatizadas. Los objetivos principales de MDA son la portabilidad, la interoperabilidad y la reutilización a través de la separación del dominio de negocio y el dominio de la plataforma de implementación [9]. Se basa principalmente en utilizar los modelos como entidades de primera clase, la separación de los conceptos o puntos de vista en dos dominios (PIM y PSM), y utilizar transformaciones entre modelos para automatizar la generación de estos [10].

MDA plantea nuevos retos que deben ser enfrentados por las empresas y equipos de desarrollo que desean utilizar este nuevo paradigma. Podemos enmarcar estos retos principalmente en 3 categorías: 1) la construcción de los meta modelos y de las transformaciones, 2) el proceso de producción de aplicaciones utilizando los metamodelos y las transformaciones, y 3) la tecnología por utilizar. En las siguientes secciones explicamos en más detalle estos retos.

2.1. Modelos y Transformaciones

MDA plantea un esquema donde se divide el mundo del problema en dos dominios, uno del negocio y otro de la plataforma. En el primero se concentran todos los conceptos del mundo del negocio, excluyendo todos los conceptos de plataforma y tecnología. Y en el otro se encuentran los conceptos de la plataforma en donde se definen todos los elementos tecnológicos necesarios para crear el sistema deseado [5].

Se crea un modelo utilizando los conceptos de mundo del negocio. Este modelo se conoce como PIM (Plataform Idependent Model) y como se dijo anteriormente, no contiene ningún concepto de la plataforma tecnológica donde se quiere implementar el sistema. Por otro lado, se define un modelo de la plataforma tecnológica, conocido como PM (Plataform Model), el cual básicamente es una representación de ésta [5].

Estos dos modelos se combinan usando una transformación automática que genera un modelo que se conoce como el PSM (Plataform Specific Model) y es un modelo del sistema que combina los conceptos del dominio del negocio y del dominio de la plataforma [5]. Esto se muestra en la figura 1.

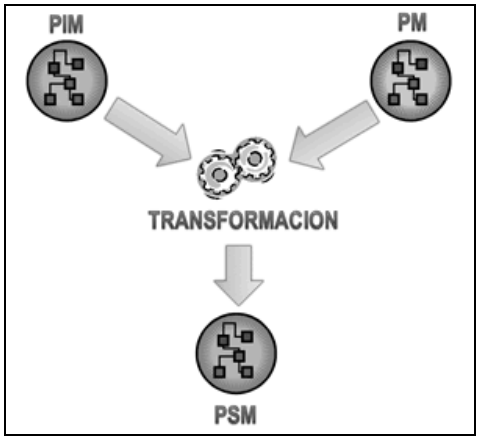


Figura 1 - Estrategia MDA

Al modelo resultado (PSM en la figura) se le aplica una generación utilizando plantillas para generar el código de la aplicación deseada.

En esta transformación se efectúa un mapeo y una composición de los elementos del dominio del negocio y del dominio de la plataforma. Se define como los elementos del negocio se combinan con uno o más elementos de la plataforma para producir uno o más elementos del PSM.

Como se ve, éste es el paso más crítico de todo el proceso MDA, y es donde se componen una gran cantidad de conceptos de ambos dominios, estructurando el modelo de la aplicación deseada.

Este planteamiento es un poco simplificador en su realización ya que pretende componer todos los conceptos de un sistema de información por medio de una única transformación de los elementos de dos dominios. Esto quiere decir que en esta transformación debemos incluir una gran cantidad de decisiones de diseño que estructurarán el producto final, lo cual reduce la flexibilidad de la solución y hace que el sueño de la multiplataforma sea muy complejo.

La complejidad de esta composición hace que sea muy difícil aplicar variabilidad a la transformación, disminuyendo la flexibilidad de la solución. Esto quiere decir que agregar, cambiar o eliminar funcionalidades y características del producto es un proceso difícil y que difícilmente puede ser apoyado por la transformación. Por otro lado, si se pretende tener varias plataformas objetivo, es necesario crear transformaciones completamente nuevas e igual de complejas, dificultando lograr la generación hacia múltiples plataformas.

2.2. El Proceso MDA

La especificación de MDA [4] no incluye un proceso definido para la creación de los metamodelos, modelos, transformaciones y otros elementos de implementación.

Tampoco es claro cómo producir aplicaciones utilizando el paradigma MDA. Esto ha generado que cada implementación de MDA desarrolle sus procesos de construcción basado en su interpretación de la especificación y sus especialidades.

Esta falta de uniformidad en la interpretación de la especificación y sus procesos hace por ejemplo que herramientas como ArcStyler [7] no cuenten con un PSM y sea necesario anotar el PIM con elementos de plataforma para la generación del código de la aplicación, lo cual contamina el PIM y resta flexibilidad a la solución. Por esto, se vuelve necesario definir un proceso que busque generalizar el desarrollo de los activos de desarrollo, y también el desarrollo de aplicaciones usando nuestra solución MDA.

2.3. Herramientas MDA

La gran mayoría de herramientas que buscan implementar MDA son desarrolladas dentro de proyectos de investigación y algunas pocas por empresas comerciales. Esto ha hecho que gran parte de estas herramientas estén incompletas, en desarrollo o hayan sido abandonadas al finalizar la investigación específica [11].

Entre las herramientas que tienen la madurez suficiente para ser utilizadas efectivamente en el desarrollo de aplicaciones podemos encontrar de dos tipos. En el primero, están las que ofrecen una solución completa al planteamiento MDA y que interpretan este planteamiento desde su propio punto de vista debido a los grandes vacíos de la especificación, como por ejemplo OptimalJ [12] o ArcStyler [7]. Estas herramientas aunque son una solución completa a la implementación de algunos conceptos de MDA, presentan una serie de inconvenientes como, por ejemplo, la anotación de elementos concretos de la plataforma en el modelaje del PIM, lo que va en contra del planteamiento de MDA, el manejo de modelos de plataforma (PM) implícitos, que no pueden ser visualizados ni manipulados, no soportan la definición de metamodelos para el PIM y el PSM lo que reduce su expresividad. Estos son algunos de los problemas que hacen que estas soluciones sean poco flexibles y con extensibilidad limitada, obligando en la mayoría de ocasiones a implementar la aplicación deseada utilizando la arquitectura y las decisiones de diseño definidas por la herramienta utilizada.

Por otro lado, la gran mayoría de herramientas MDA se encuentran en el segundo grupo y se enfocan en resolver problemas específicos de MDA como el modelaje, las transformaciones o la generación de código. Entre este tipo podemos encontrar varias que logran resolver cada uno de los problemas de MDA de forma flexible y muy funcional. Estas herramientas han logrado llegar a un nivel de especialización interesante y que nos hace desear contar con una solución que integre las mejores funcionalidades de cada una.

En los siguientes capítulos presentamos el caso de estudio y mostramos las decisiones que tomamos para remediar cada uno de los retos mencionados.

3. Caso de Estudio

El caso de estudio consistió en desarrollar un sistema de administración de configuraciones de software de alto nivel denominado *Changeset. Changeset* debe administrar principalmente conceptos de proyectos, solicitud de cambio, ítems de configuración, versiones y líneas de base. Sobre estos conceptos se debe construir toda la funcionalidad de la aplicación.

Esta aplicación pertenece a un conjunto restringido de aplicaciones MIS (Management Information Systems) [13], las cuales utilizan las propiedades CRUD (Create, Retreive, Update y Delete) [14].

El alcance de este proyecto se definió para producir la capa de lógica de la aplicación, dejando la interfaz usuario para trabajos futuros.

3.1. Estrategia Global

Para llegar desde el modelo de negocio al código de la aplicación sobre una plataforma específica, nuestra estrategia consiste en realizar primero una transformación a un modelo de arquitectura. Este modelo contiene elementos del diseño arquitectónico de la aplicación independientes de la plataforma. A partir de el diseño de la arquitectura de la aplicación trasformamos a un modelo de una plataforma específica. Por último, se transforma al modelo del lenguaje donde se representa la implementación de los conceptos de la plataforma.

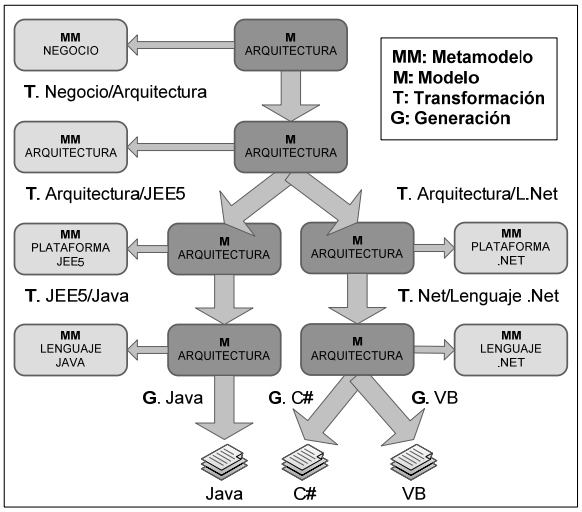


Figura 2 - Estrategia Propuesta

Para realizar las transformaciones creamos cuatro metamodelos: 1) de negocio, 2) de arquitectura, 3) de plataforma y 4) de lenguaje. La figura 2 muestra como el modelo de entrada se va transformando en modelos que se van refinando hasta llegar al código.

Esta estrategia se puede ver en la figura 2, donde se muestran los diferentes modelos, metamodelos, transformaciones y la generación para cada una de las plataformas objetivo.

En las siguientes secciones presentamos los conceptos de los distintos metamodelos y de lo que se espera produzca cada una de las transformaciones. Estas secciones son un resumen de las ideas generales, desarrolladas durante la investigación de trabajo de grado de maestría que se encuentran en [15, 16].

3.2. El Metamodelo de Negocio

El modelo de negocio se debe construir únicamente teniendo en cuenta los conceptos del problema y del tipo de aplicación que se quiere desarrollar. De esta forma, el primer paso en nuestra estrategia fue desarrollar un metamodelo que contuviera conceptos propios de un subconjunto de aplicaciones MIS. Los principales elementos que constituyen este metamodelo son:

- *BusinessEntities:* Estos elementos representan una entidad del negocio que administra otras entidades o elementos.
- *BusinessElements:* Concepto del negocio que es administrado por una entidad.
- *Relationships:* Relaciones entre las entidades y los elementos del negocio. Pueden ser *Single* cuando una entidad usa otra entidad o elemento, o *Multiple* cuando una entidad administra una colección de una o varias entidades o elementos del negocio.
- *Contracts:* Este concepto representa los servicios que las entidades y elementos del negocio van a ofrecer. Los contratos de *Create, Detail, Update, Delete*, y *List* están caracterizados especialmente.

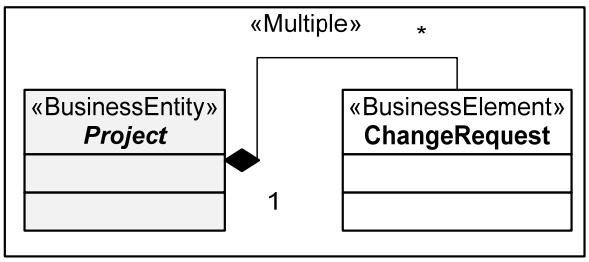


Figura 3 - Modelo de Changeset

La figura 3 muestra algunos de los elementos del modelo de negocio de Changeset marcados con los

estereotipos del metamodelo de negocio presentado. En esta figura vemos como la entidad de negocio *Project* administra una colección de *ChangeRequest* por medio de una relación *Multiple.* Esto representa el conjunto de solicitudes de cambio que se han realizado sobre un proyecto.

3.3. Metamodelo de la Arquitectura

En nuestra solución, construimos un metamodelo de Arquitectura que nos permite representar una arquitectura multicapas y patrones de diseño independiente de plataforma. Los principales elementos que constituyen este metamodelo son:

- *Layers:* Estos elementos representan las capas de las entidades y elementos de una arquitectura multicapa, estas están especializadas en *Presentación, Aplicación. Servicios del sistema, Servicios del negocio, Persistencia y Almacenamiento.*
- *Communications*: Este representa la comunicación que existe entre las diferentes capas de un mismo concepto del negocio.
- *Associations*: esta es la relación existente entre las capas de diferentes conceptos. Se especializan en *simple* y *multiple* según el numero de capas relacionadas.
- *ValueObject*: concepto que se utiliza para la comunicación y asociación entre las diferentes capas.

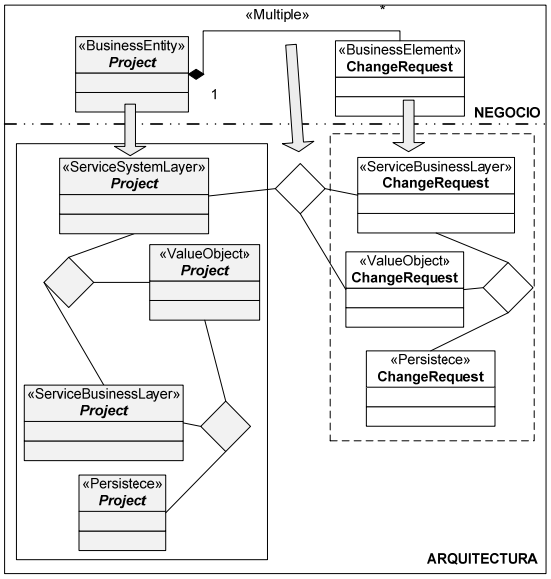


Figura 4 - Mapeo Negocio-Arquitectura

Con estos elementos se representan los conceptos del negocio más los conceptos arquitectónicos de una arquitectura multicapas. Por ejemplo, la entidad de negocio *Project* en este dominio se representa por medio de un conjunto de capas. Estas capas son: la capa de servicios del sistema, la capa de servicios del negocio, la capa de persistencia. Además de estas capas se representan los conceptos de comunicación entre ellas, las asociación de la capa de servicios de sistema de *Project* con la capa de servicios del negocio de *ChangeRequest* y los *ValueObject* de *Project* y *ChangeRequest* para el transporte de datos entre las capas. La figura 4 muestra el mapeo que se realiza entre la entidad de negocio *Project* y las capas en el modelo de arquitectura.

3.4. Metamodelo de la Plataforma

Siguiendo la misma estrategia, creamos un metamodelo para cada una de las dos plataformas objetivo JEE5 y NET.

Los principales elementos que constituyen el metamodelo JEE5 son:

- *EJBDeploy:* Elemento que permite el encapsulamiento de varios componentes para el deploy de la aplicación.
- *EnterpriseBean:* Componente EJB que se especializa en *Entity* y *Session.*
- *Comunicación*: Permite la comunicación entre los beans
- *DAOObject:* elemento de acceso a datos utilizado por los beans de entidad.

Por otro lado, los principales elementos que constituyen el metamodelo .NET son:

- *NetDeploy:* Elemento que permite el encapsulamiento de varios componentes para el deploy de la aplicación.
- *EnterpriseComponent:* Componente empresarial que será publicado en el servidor. Se especializa en *Business Component* y *Administrable Component* Estos últimos son administrados por el servidor.
- *Comunicación*: Permite la comunicación entre los componentes
- *DAOObject:* elemento de acceso a datos utilizado por los componentes

En el caso de JEE5 tenemos un *Session* para la capa de servicios de sistema de *Project*, un *Entity* para la capa de servicios del negocio de *Project* y otro para la de *ChangeRequest*, y un *DAOObject* para la capa de persistencia de *Project* y otro para *ChangeRequest.* La figura 5 muestra el resultado de la transformación para JEE5.

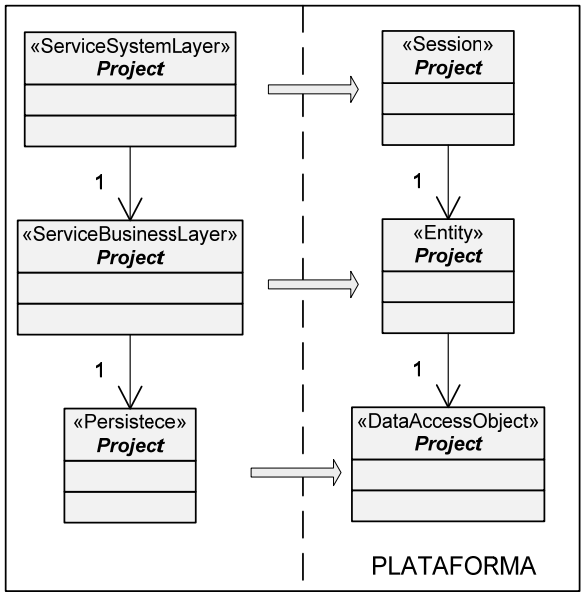


Figura 5 - Mapeo Arquitectura - JEE5

3.5. Metamodelo del Lenguaje

El metamodelo del lenguaje contiene conceptos como clases, interfaces, métodos, etc. El primer metamodelo es el de Java y es utilizado para implementar soluciones en JEE5. El metamodelo de Lenguaje .NET, permite para generar aplicaciones en diversos lenguajes como C# y Visual Basic.

Los principales elementos que constituyen el metamodelo Java son:

- *JavaClass:* Elemento que representa una clase Java
- *JavaInterface:* Interface Java
- *Package:* Elemento contenedor de clases e interfaces Java
- *Metodos:* Este concepto representa los métodos que los beans van a ofrecer. Los métodos de *Create, Detail, Update, Delete*, y *List* están caracterizados especialmente.

Los principales elementos que constituyen el metamodelo .NET son:

- *NetClass:* Elemento que representa una clase Java
- *NetInterface:* Interface Java
- *Project:* Elemento que empaqueta un grupo de elementos que serán publicados con un componente.
- *Metodos:* Este concepto representa los métodos que los beans van a ofrecer. Los métodos de *Create, Detail, Update, Delete*, y *List* están caracterizados especialmente.

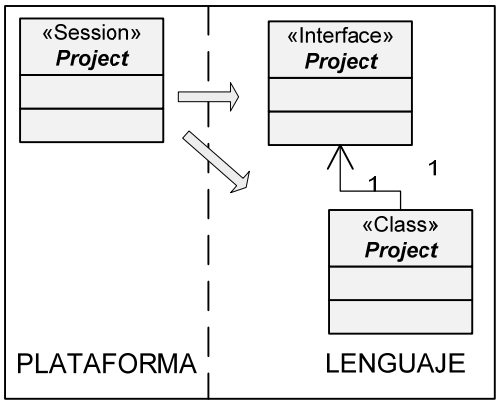


Figura 6 - Mapeo Plataforma - Lenguaje

La figura 6 muestra como el elemento *Session Project* del modelo de arquitectura, se transforma en una clase y una interfaz que lo implementan.

4. Las Transformaciones en el caso de estudio

Dado que tenemos cuatro dominios: negocio, arquitectura, plataforma y lenguaje, significa que se requieren varias transformaciones entre estos dominios, y una generación de código al lenguaje en el que se quiere implementar la aplicación. Ver figura 2.

En cada una de estas transformaciones, se hace un mapeo de los conceptos del metamodelo de origen con los conceptos del modelo destino. Además, se define cómo estos conceptos se van a componer en el modelo generado.

4.1. Consideraciones generales en las transformaciones

Las transformaciones de modelo a modelo las expresamos utilizando un lenguaje llamado ATL [6]. Este lenguaje admite expresar las reglas de transformación de manera declarativa e imperativa. En este caso de estudio, todas las reglas de transformación fueron expresadas de manera declarativa.

Una regla básica está compuesta de los elementos presentados en la figura 7. Los elementos de entrada pertenecen al metamodelo origen y los de salida al metamodelo destino. Para acceder a los elementos de entrada, ATL utiliza expresiones OCL [17] y para crear elementos del metamodelo de salida, se cuenta con constructores del lenguaje que permiten crear,

modificar y asociar valores a los objetos, atributos y relaciones.

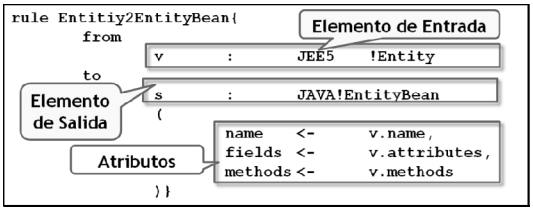


Figura 7 - Regla de Transformación

Durante el desarrollo del caso de estudio, se identificaron una serie de patrones de transformación que facilitaron la construcción de las distintas transformaciones. Estos patrones están relacionados con los distintos tipos de mapeos que se necesita entre los elementos de entrada y los de salida. Algunas son transformaciones simples elemento a elemento, pero otras requieren que a partir de uno (o más) de entrada se produzca uno (o más) de salida.

En las siguientes secciones presentamos de manera breve lo que se construye en cada transformación.

4.2. Transformación de negocio a arquitectura

En esta transformación se estructura la aplicación en diferentes capas. Es aquí donde una entidad de negocio se convierte en cuatro elementos diferentes, un elementos de capa de servicios del sistema que expone los servicios de esta entidad y de los elementos que ésta administra, un elemento de servicios del negocio donde están los servicios que esta entidad expone, un elemento de persistencia el cual expone los servicios de acceso y modificación de la persistencia y un *ValueObject* que será utilizado para la comunicación entre las diferentes capas de la entidad y con la asociación con las capas de otras entidades. Los elementos de negocio solamente se transforman en la capa de servicios de negocio, la capa de persistencia y en el *ValueObject*. Esta transformación también se encarga de establecer la comunicación entre las diferentes capas y la asociación con las capas de otras entidades o elementos utilizando un *ValueObject*, como elemento de transporte de datos. Ver figura 4.

4.3. Transformación de arquitectura a plataforma

Aquí se toman los conceptos de cada capa y se define cuál servicio del contenedor será el encargado de implementarlo, utilizando una transformación para JEE5 y otra para .NET. Estas transformaciones se ocupan principalmente de convertir las capas de servicios del sistema en *Session*, servicios del negocio en *Entitys* y persistencia en *DAOObject* para JEE5. Y en el caso de .NET para las capas de servicios del sistema se producen *Administrable Components*, para servicios del negocio se producen *Business Components* y para persistencia se producen *DAOObjects*. Estas transformaciones además deben mantener las diferentes comunicaciones entre los diferentes beans y componentes. Ver figura 5.

4.4. Transformación de plataforma a lenguaje

Esta transformación es la que se encarga de construir la implementación de cada *EnterpriseBean* o *Enterprise Component,* según el caso, con las interfaces, clases, métodos y atributos necesarios, manteniendo la comunicación entre ellos y empaquetándolos según el esquema de publicación deseado. Como en el caso de la plataforma existen dos transformaciones, una para JEE5 y otra para .NET. En nuestro caso por ejemplo un *Session* transforma en una clase y una interfaz Java. Ver figura 6.

4.5. Generación de Código (Java, VB, C#)

Después de aplicar las tres transformaciones entre modelos, el siguiente paso es convertir estos modelos en código. Para esto se requiere recorrer los elementos del modelo del sistema en el dominio del lenguaje y combinarlos con plantillas que generen el código de la aplicación. Parte del código de la aplicación es generado pero hay otra porción que debe ser escrita manualmente por los desarrolladores. Esto se debe a que la capacidad de expresión de los metamodelos que se usan, tiene un alcance limitado, permitiéndonos crear únicamente los cuerpos de los métodos estándar (create, update, list, detail, delete y los métodos que permiten el manejo de relaciones entre entidades). En cambio para los métodos que implementan las reglas del negocio y funcionalidades especiales, solamente se genera el esqueleto de estos, dejando que esta labor sea llevada acabo por los desarrolladores. La herramienta que se usó garantiza

la protección de este código no generado, así que si se modifican los modelos y se regenera el código, las modificaciones no se pierden.

5. Implementación

Para la implementación de nuestra solución se seleccionó una suite de herramientas plug-ins de Eclipse [18]. Estas se describen a continuación:

- *Una manejador de metamodelos y modelos:* esta herramienta debe permitir definir metamodelos y modelos, que sean serializados en formato XMI y seguir los estándares de MOF propuestos por la OMG. Para esto se escogió a EMF [19]. EMF implementa EMOF (Essential MOF), el cual es el subconjunto básico de MOF.
- *Una herramienta gráfica de modelaje:* Debido a la complejidad en la expresión de los metamodelos se requiere de una herramienta que permita modelar gráficamente. Se escogió GMF como herramienta de modelado. Esta herramienta permite definir metamodelos gráficamente y generar un plug-in que permite crear los modelos conformes a dicho metamodelo [20].
- *Un lenguaje de transformación de modelo a modelo:* Un lenguaje que permita definir transformaciones entre los metamodelos de dos dominios. Se seleccionó ATL como lenguaje de transformación, por ser un lenguaje mixto (Declarativo e imperativo), compatible con modelos en expresados con EMF [6] y tener una implementación consistente, aun cuando no es una implementación de QVT (Query, View, Transformations), el estándar propuesto por la OMG para hacer consultas y transformaciones sobre los modelos y metamodelos.
- *Un lenguaje de transformación de modelo a código:* Se requiere un lenguaje de transformación que se especialice en la generación de archivos de texto (código, XML, documentación) a partir de modelos. Usamos Acceleo debido a poder recibir modelos expresados con EMF, utilizar plantillas para la generación y poder manejar adecuadamente código no generado [21].

6. Conclusiones y trabajo futuro

Durante la implementación del sistema de administración de configuraciones *Changeset*, utilizando el paradigma de desarrollo orientado a modelos, encontramos varios elementos que vale la pena resaltarse.

Una de las lecciones más importantes para la implementación de MDA en el desarrollo de aplicaciones empresariales, es que no se puede obtener un modelo de aplicación sobre una plataforma específica sólo con el conocimiento del negocio. La transformación que se requeriría para proceder de esta manera tendría una complejidad muy alta y carecería de flexibilidad. Una posible solución, planteada en este artículo, consiste en ir refinando el modelo de negocio a través de varias transformaciones para ir incluyendo conceptos de arquitectura, plataforma y lenguaje. De manera secuencial, se van construyendo todos los elementos necesarios para llevarlo a código. Las transformaciones que se utilizan para esto son más simples y flexibles, lo cual nos obliga a hacer un análisis que nos permita identificar los conceptos de arquitectura y de plataforma que se utilizarán en el desarrollo.

La definición de los metamodelos de cada uno de los dominios usados en el desarrollo es un proceso que requiere análisis profundo, pero además se debe retroalimentar de la creación de modelos que validen sus conceptos y también de la construcción de transformaciones. En esta construcción se validan los conceptos expuestos en el metamodelo y se verifica que su expresividad sea la adecuada para generar un modelo en el nuevo dominio y finalmente el código de la aplicación.

Este trabajo ejemplifica la implementación de MDA en el mundo real, y además los metamodelos y transformaciones desarrollados permiten generar nuevas aplicaciones, que se enmarquen dentro del subconjunto de aplicaciones MIS definido anteriormente. Lo cual permite aumentar la productividad ya que la infraestructura desarrollada puede ser reutilizada, minimizando los costos de futuros desarrollos.

Como trabajos futuros, queremos utilizar nuestra estrategia con otras aplicaciones de negocio para poder encontrar elementos generales que permitan definir un proceso claro de construcción de los artefactos como metamodelos, transformaciones, modelos y plantillas. La idea última es construir una línea de producto basada en modelos que permita mejorar la productividad de los equipos de desarrollo.

7. Referencias

[1] D. C. Schmidt, "Model-Driven Engineering," in *Computer*. vol. 39, 2006, pp. 25-31.

[2] J. Greenfield and K. Short, *Software Factories*: Wiley Publishing, 2004.
[3] OMG, *Model Driven Architecture*,[En Linea], Disponible: http://www.omg.org/mda/ Ultima visita: Octubre 2006.
[4] J. Mukerji and J. Miller, *MDA Guide*,[En Linea], Disponible: http://www.omg.org/docs/omg/03-06-01.pdf Ultima visita.
[5] S. J. Mellor, K. Scott, A. Uhl, and D. Weise, *MDA Distilled, Principles of Model-Driven Architecture*: Addison Wesley Professional, 2004.
[6] F. Jouault and I. Kurtev, "Transforming Models with ATL," *Lecture Notes in Computer Science,* vol. Volume 3844/2006, pp. 128-138, 2006.
[7] I. Objects, *ArcStyler*,[En Linea], Disponible: http://www.interactive-objects.com/products/arcstyler Ultima visita: Octubre 2006.
[8] Microsoft, *Domain-Specific Language Tools*,[En Linea], Disponible: http://msdn.microsoft.com/vstudio/DSLTools/ Ultima visita: Octubre 2006.
[9] F. Truyen, "The Fast Guide to Model Driven Architecture," 2006.
[10] K. Lano, *Advanced System Design with Java , UML and MDA*: Butterworth-Heinemann, 2005.
[11] PlanetMDE, *Open Source Tools and Research Tools*,[En Linea], Disponible: http://www.planetmde.org/tools/index.html Ultima visita: Octubre 2006.
[12] Compuware, *OptimalJ*,[En Linea], Disponible: http://www.compuware.com/products/optimalj/ Ultima visita: Octubre 2006.
[13] D. Duffy, *Domain Architectures - Models and Architectures for UML Applications*: John Wiley&Sons, 2004.
[14] S. Vandivier and K. Cox, *Oracle Application Server*: Oracle Press, 2002.
[15] J. Bohorquez, "Framework De MDA para aplicaciones empresariales hacia plataformas J2EE Y .Net - Metamodelos," in *Grupo de Construcción de Software* Bogota: Universidad de los Andes, 2006.
[16] A. Yie, "Framework De MDA para aplicaciones empresariales hacia plataformas J2EE Y .Net - Transformaciones," in *Grupo de Construcción de Software* Bogota: Universidad de los Andes, 2006.
[17] OMG, "OCL 2.0 - OMG Final Adopted Specification," 2003.
[18] PlanetMDE, *Open Source Tools and Research Tools*,[En Linea], Disponible: http://www.planetmde.org/tools/index.html Ultima visita: 2006.
[19] Wikipedia, "CRUD Properties," 2006.
[20] *Graphical Modeling Framework* [En Linea], Disponible: http://www.eclipse.org/gmf/ Ultima visita: Octubre 2006.
[21] S. E. I. SEI, *Software Product Lines*,[En Linea], Disponible: http://www.sei.cmu.edu/productlines/ Ultima visita: Diciembre 2006.

Evolução de um Processo Ágil de Desenvolvimento baseado em framework

Franciene Duarte Gomes*
Maria Istela Cagnin
UNIVEM - Fundação de Ensino Eurípides Soares da Rocha,
Marília, São Paulo, Brasil, Caixa Postal 2041, CEP 17525-901
franciene@gmail.com,
istela@univem.edu.br

Resumo

Buscar uma melhor qualidade tanto no processo de desenvolvimento quanto no produto de software é um dos objetivos da Engenharia de Software. Vários são os meios utilizados para isso, como exemplo tem-se: o uso de estimativas em projetos de software a fim de cumprir os prazos previstos, a utilização de métodos ágeis a fim de reduzir o tempo de desenvolvimento e permitir adaptações em qualquer fase do desenvolvimento, o uso de técnicas de reúso (frameworks, linguagens de padrões, geradores de aplicação, etc) a fim de diminuir o tempo de desenvolvimento, reutilizando partes de código, entre outros recursos. Este artigo apresenta a evolução de um processo ágil de desenvolvimento de software baseado em framework guiada por meio de um estudo de caso.

Palavras chaves: Processo de Desenvolvimento, Métodos Ágeis, *Framework,* Linguagem de Padrões.

Abstract

Searching for a better quality in the development process and in software product is one of the software engineering objectives. There are severals means for that, like the use of estimatives in software projects aiming accomplish deadlines, the use of agile methods aiming reduces the development time and allowing adaptations in a development phase, the use of reuse techniques (framework, pattern languages, application generators, etc) aiming to low the development time, reuse code parts, among other resources. This paper presents the evolution of a software development agile process based on framework driven through a case study.

Keywords: Development Process, Agile Methods, Frameworks, Pattern Language.

1. Introdução

Existem várias definições de processo de *software* [14, 15, 18] na literatura. No entanto, todas elas possuem um ponto em comum, ou seja, determinam que o processo de *software* seja formado por um conjunto de atividades que apóiam o desenvolvimento do *software.* Essas atividades abordam todo o ciclo de desenvolvimento de *software* e quando aplicadas corretamente permitem aumentar a qualidade do *software* e, conseqüentemente, sua produção.

Segundo Sommerville [18], não há um processo considerado ideal e diferentes empresas desenvolvem abordagens diferentes para o desenvolvimento de *software*. Essa afirmação está relacionada à grande quantidade de processos existentes, buscando atender a diversos tipos de *software.*

Com base nisso, os métodos ágeis [1] (como exemplo: XP [4], Scrum [1], DSDM (*Dynamic System Development Method*) [1], etc) buscam aumentar a produtividade do *software,* principalmente devido a suas principais características, ou seja, iterativos, incrementais e adaptativos. Essas características possibilitam a redução de regras existentes em processos de *software*, denominados como processos tradicionais (como exemplo: modelo seqüencial linear ou modelo cascata, modelo incremental e modelo espiral [15]).

No entanto, a evolução de um processo, de acordo com Sommerville [18], depende da capacidade das pessoas em explorar o processo e, segundo Pressman [15], a qualidade do *software* é incorporada durante o uso do processo, com a aplicação correta dos métodos, procedimentos e ferramentas nele disponível.

Neste contexto, este artigo apresenta uma evolução de um processo ágil de desenvolvimento de *software* baseado em *framework* [11], denominado PARFAIT/EA [12], cuja construção do *framework* seja baseada em um LPA (Linguagem de Padrões de Análise) [3]. Uma LPA é formada por vários padrões de *software* utilizado para resolver um determinado problema. O uso de uma LPA em um *framework*

*Apoio Financeiro da CAPES

facilita a instanciação do *framework*, o entendimento do seu domínio e gera documentação de acordo com os padrões selecionados.

Para avaliar a evolução do PARFAIT/EA bem como sua aplicabilidade, um estudo de caso de desenvolvimento de *software* foi conduzido. Os resultados obtidos e dificuldades observadas são discutidos neste artigo.

Na Seção 2 são relatados os trabalhos relacionados. Na Seção 3 apresenta-se resumidamente a versão preliminar do processo PARFAIT/EA. Na Seção 4 apresenta-se o planejamento, a execução e os resultados obtidos do estudo de caso conduzido para analisar e evoluir o processo PARFAIT/EA. Na Seção 5 apresenta-se a evolução do processo PARFAIT/EA, considerando os resultados obtidos no estudo de caso e na Seção 6 discutem-se as conclusões finais e os trabalhos futuros.

2. Trabalhos Relacionados

Alguns trabalhos relacionados a processos ágeis de desenvolvimento, mas não baseados em *framework* são discutidos nesta seção.

O XwebProcess [17] é um processo ágil para desenvolvimento de aplicações *web*, permitindo participação do cliente, descrição do projeto e aplicação de teste *web* para garantir a qualidade do produto desenvolvido. Este processo é baseado no método ágil XP e é documentado por meio do meta-modelo SPEM[1] (*Software Process Engineering Metamodel*), que é utilizado para descrever um processo de desenvolvimento de *software*.

Ao contrário do processo PARFAIT/EA, que é baseado em *framework,* o processo XwebProcess não é baseado em nenhuma tecnologia ou ferramenta específica. Alguns resultados comparados com o método XP e XWebProcess mostra que este último produz documentação suficiente e favorece a comunicação em equipe [17].

Um outro processo ágil de desenvolvimento de *software* encontrado na literatura é o easYProcess [10], que é apoiado por práticas ágeis dos métodos RUP (*Rational Unified Process*) [16], XP [4] e Modelagem Ágil [2]. Foi desenvolvido em ambiente acadêmico com o objetivo de facilitar o aprendizado da disciplina de Engenharia de Software. As características principais do processo easYProcess [10] são: participação do cliente, desempenho de diversos papéis por uma única pessoa, liberação de *releases* e realização de iterações curtas de acordo com o ambiente acadêmico, aplicação de testes durante o projeto e de práticas ágeis e criação de repositório de código para controle de versão. O easYProcess também é denominado de YP[2] e utiliza várias ferramentas livres como exemplo: Xplanner, utilizada para gerência de projetos, JUnit, para implementação de teste de unidade, entre outras.

3. PARFAIT/EA

Um esboço do processo PARFAIT/EA foi elaborado a partir da abstração de um processo ágil de reengenharia baseado em *framework*, denominado PARFAIT [7], que migra sistemas legados procedimentais para o paradigma orientado a objetos. Isso foi possível por meio da análise de um estudo de caso de reengenharia conduzido pelo processo PARFAIT, em que algumas atividades do PARFAIT foram mantidas, outras removidas, alteradas e adicionadas para atender o desenvolvimento.

Similarmente ao processo PARFAIT o processo PARFAIT/EA é dividido em quatro fases, sendo: CONCEPÇÃO, ELABORAÇÃO, CONSTRUÇÃO e TRANSIÇÃO. Cada fase do processo é composta por atividades, que são utilizadas para apoiar cada etapa do desenvolvimento do *software.*

Nesta versão preliminar do processo PARFAIT/EA não foram definidas diretrizes para apoiar na execução das atividades e nem inspeções para avaliar os artefatos produzidos em cada atividade.

O processo PARFAIT/EA é definido como um processo ágil, pois utiliza em suas atividades práticas ágeis de alguns métodos ágeis. Além disso, incorpora as características iterativo, incremental e adaptativo dos métodos ágeis. Uma outra característica do PARFAIT/EA, que colabora com a sua agilidade, é o uso de *frameworks* de aplicação [11], baseados em linguagens de padrões, que apóiam a construção automática do *software* e liberam uma versão funcionando em cada iteração do processo.

As práticas ágeis utilizadas pelo PARFAIT (metáfora, jogo do planejamento, cliente presente, testes constantes, programação em pares, propriedade coletiva do código, versões pequenas e integração contínua) foram mantidas na versão inicial do PARFAIT/EA.

Na Figura 1 apresenta-se uma visão geral do processo PARFAIT/EA, contendo as suas fases e atividades. Todas as atividades do processo são descritas por passos numerados para facilitar sua execução, mas não estão mostrados na Figura 1 por falta de espaço.

[1] http://www.omg.org/technology/documents/formal/spem.htm

[2] http://www.dsc.ufcg.edu.br/~yp/

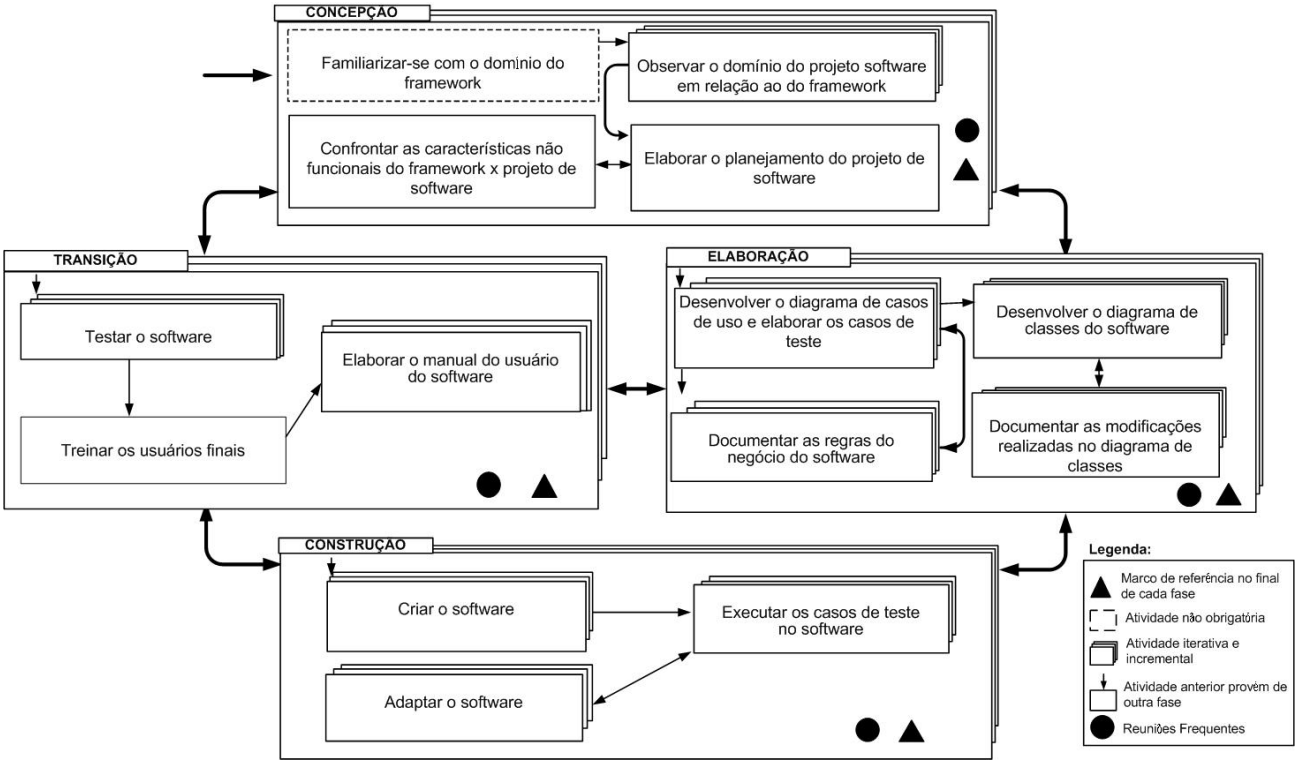


Figura 1 – Versão simplificada do processo PARFAIT/EA (adaptada de [12])

Em todas as fases do processo foram definidos marcos de referências, representados na Figura 1 por um triângulo, e a atividade "Reuniões Freqüentes", representada por um círculo. O objetivo do uso de marcos de referência é avaliar se a execução de cada atividade da fase foi aplicada corretamente e verificar como está o andamento do projeto de *software*. As reuniões freqüentes permitem revisar o que já foi realizado e o que necessita ser feito. O objetivo principal desta atividade é a participação do cliente, podendo ser realizada no início ou no fim da fase ou na conclusão de cada atividade. Essa atividade não existia no processo PARFAIT e foi adicionada no PARFAIT/EA.

A classificação das atividades do PARFAIT/EA segue a mesma definição do PARFAIT [7], sendo classificadas como: obrigatória, não obrigatória, iterativa e incremental. Segundo Gomes *et al.* [12], essa classificação é importante pois orienta o engenheiro de *software* na execução das atividades e na seqüência em que devem ser executadas.

As atividades iterativas e incrementais são representadas na Figura 1 por um retângulo de linha dupla. Nessas atividades, o engenheiro de *software* pode reexecutá-las a fim de refinar os artefatos anteriormente elaborados.

A primeira fase do processo PARFAIT/EA é a fase de CONCEPÇÃO. O objetivo principal dessa fase é identificar o domínio do projeto de *software* com o do *framework,* verificar se os requisitos não funcionais do projeto de *software* são atendidos pelo *framework* e, caso não seja, avaliar a viabilidade em continuar ou não o projeto de *software* e elaborar o planejamento do projeto de *software* baseado em projetos anteriores. As atividades desta fase são: "Familiarizar com o domínio do framework", "Observar o domínio do projeto de software em relação ao framework", "Elaborar o planejamento do projeto de software" e "Confrontar as características não funcionais do framework x projeto de software".

A fase seguinte é a de ELABORAÇÃO. Esta fase tem como objetivo criar a documentação do projeto de *software*, principalmente com o apoio de uma linguagem de padrões de análise, de acordo com os ciclos de desenvolvimento. As atividades desta fase são: "Desenvolver o diagrama de casos de uso e elaborar os casos de teste", "Documentar as regras de negócio do software", "Desenvolver o diagrama de classes do software" e "Documentar as modificações realizadas no diagrama de classes".

A próxima fase é a de CONSTRUÇÃO, sendo que é nesta fase que o *software* é criado, por meio da instanciação do *framework* e os casos de testes, elaborados na fase anterior, são aplicados no *software* para garantir a sua qualidade.

Ainda na fase de CONSTRUÇÃO, todas as adaptações solicitadas pelo cliente necessárias ao projeto de *software* são executadas. As adaptações no

software também ocorrem quando os requisitos do *software* não são fornecidos pelo *framework* ou são fornecidos pelo *framework*, mas não estão presentes no projeto de *software*. Esta fase é composta pelas atividades: "Criar o software", "Executar os casos de teste no software" e "Adaptar o software".

Após o *software* passar pela fase de CONSTRUÇÃO, a próxima e última fase do processo é a fase de TRANSIÇÃO. O objetivo da fase de TRANSIÇÃO é testar o *software* juntamente com o cliente, verificando se o que foi proposto até o momento foi incorporado ao *software.* A realização de treinamentos com o usuário e a elaboração do manual do usuário são executadas nesta fase. Para a versão preliminar do processo PARFAIT/EA, ficou definido que a atividade "Converter a base de dados do software" ficaria como uma atividade isolada, sendo utilizada somente quando houvesse a necessidade por parte do cliente e com uma avaliação realizada pela equipe de projeto para verificar a possibilidade da conversão de dados existentes. As atividades da fase de TRANSIÇÃO são: "Testar o software", "Treinar os usuários finais", "Elaborar o manual do usuário do software" e "Converter a base de dados do software".

Com base nesta versão preliminar do processo PARFAIT/EA, um estudo de caso foi planejado e conduzido, a fim de evoluir e aprimorar tal processo e é apresentado na próxima seção.

4. Estudo de Caso Planejado

Para avaliação do processo PARFAIT/EA, um planejamento de um estudo de caso, foi definido e documentado de acordo com Wholin [19]. Esse planejamento tem como objetivo observar os resultados da aplicação do processo PARAFAIT/EA e sua evolução no contexto de desenvolvimento de *software*.

4.1 Definição

Esta fase do planejamento tem como objetivo evidenciar o que realmente pretende-se avaliar, estabelecendo as possíveis hipóteses para analisar o processo e avaliar sua aplicabilidade no desenvolvimento de *software.*

Objeto de estudo: PARFAIT/EA - Processo Ágil de Desenvolvimento baseado em *Framework*.

Objetivo: Conduzir um estudo de caso no contexto de desenvolvimento de *software* para avaliar o uso do processo PARFAIT/EA.

Foco qualitativo: Produção de documentação necessária para apoiar na produção do *software* que atenda as necessidades dos usuários e clientes, seguindo várias práticas de métodos ágeis e apoio computacional baseado em *framework*.

Perspectiva: A perspectiva do estudo de caso baseia-se no uso do processo pelo engenheiro de *software*.

Contexto: O estudo de caso foi realizado pela própria autora do processo o qual já havia executado uma análise de um estudo de caso de reengenharia para a definição das fases e atividades do processo PARFAIT/EA [12]. O material necessário para o estudo foi: documentação da LPA GRN (Gestão de Recursos e Negócios) [5], documentação do *framework* GREN [5], da ferramenta de instanciação GREN-Wizard [6], da ferramenta de controle de versões GREN-WizardVersionControl [8], manuais sobre o SGBD MySQL[2], sobre a linguagem de programação Smalltalk. O estudo de caso baseia-se em uma empresa real, no ramo de prestação de serviços. O *software* desenvolvido controla os serviços e produtos oferecidos por uma empresa incubadora[3] às empresas incubadas. Alguns dos serviços oferecidos por essa empresa são: cópias de xérox e envio de fax, bem como empréstimos de revistas de informática, administração, negócios entre outras.

4.2 Planejamento

Seleção do Contexto: Para a condução do estudo de caso, um formulário foi fornecido para anotar o tempo gasto na execução de cada atividade e de cada ciclo de desenvolvimento. A autora do processo já possuía algum conhecimento nos recursos e ferramentas necessárias para a condução do estudo, adquirido em um treinamento. Esse estudo é válido em um contexto específico do domínio de Engenharia de Software.

4.3 Formulação das hipóteses:

Nulas:

1. A documentação produzida pelo processo para a instanciação do *framework* e para adaptação da versão gerada pelo *framework* é suficiente para apoiar o desenvolvimento de *software*.
2. Na definição da lista de requisitos, a maior parte das regras de negócios é identificada.
3. Todas as práticas ágeis definidas em cada atividade do processo PARFAIT/EA são utilizadas em todas as iterações do processo.

[2] http://dev.mysql.com/doc/refman/4.1/pt/

[3] Empresa deste tipo tem como objetivo oferecer um ambiente flexível e facilidades em recursos como: qualificação, infra-estrutura, serviços básicos, como por exemplo: telefonia e acesso *web.*

4.4 Seleção das variáveis

As variáveis independentes são variáveis que podem ser manipuladas e controladas e as variáveis dependentes são as variáveis que se observa o efeito das mudanças das variáveis independentes [19].

4.4.1 Variáveis independentes: A experiência do participante com os recursos necessários para a execução do estudo de caso (como: o uso do *framework* GREN, da linguagem de padrões GRN, da ferramenta de instanciação GREN-Wizard e da ferramenta de controle de versões GREN-WizardVersionControl, do SGBD MySQL, da linguagem de programação Smalltalk, dos critérios de teste funcional: Particionamento de Equivalência e Análise do Valor Limite) e no domínio de *softwares* de prestação de serviços ajuda no desempenho e no uso do processo.

4.4.2 Variáveis dependentes: 1) analisar se a lista de padrões de análise, o diagrama de classes e a lista de requisitos são suficientes tanto para apoiar a instanciação do *framework* quanto para apoiar as adaptações no *software* gerado, a fim de adequá-lo ao projeto de *software* definido na fase de CONCEPÇÃO; 2) quantidade de regras de negócio identificada em cada ciclo de desenvolvimento durante a definição da lista de requisitos; e 3) quantidade e identificação de práticas ágeis utilizadas em cada iteração do processo.

A primeira variável baseia-se na documentação produzida, verificando se essa documentação é suficiente para um projeto de *software*, a segunda variável relaciona-se na identificação de regras de negócio com a liberação de versões funcionais ao cliente e a terceira variável baseia-se na freqüência de uso de práticas ágeis em cada iteração do processo.

4.4.4 Seleção dos indivíduos: A técnica utilizada para selecionar e executar o estudo de caso é a amostragem por conveniência, executada pela própria autora do processo.

4.4.5 Projeto do estudo de caso: O estudo de caso é definido como objeto único, pois foi realizado por apenas um participante em um único estudo de caso.

4.4.6 Instrumentação: Para a condução do estudo de caso é necessário que o participante possua conhecimento em algumas técnicas. Alguns documentos são utilizados para treinamento, como: documentação do processo PARFAIT/EA [12], documentação da LPA GRN e do *framework* GREN, manual da linguagem de programação Smalltalk, documentação dos critérios de teste funcionais Particionamento de Equivalência e Análise do Valor Limite, da ferramenta de instanciação GREN-Wizard e da ferramenta para controle de versões GREN-WizardVersionControl.

4.4.7 Avaliação da validade: Como ameaça à avaliação dos resultados tem-se: a validade de conclusão em que algumas informações e recursos estabelecidos dependem do conhecimento do engenheiro de *software*; a validade interna em que o estudo de caso foi realizado pela autora do processo, que já possui experiência nos recursos necessários pelo processo PARFAIT/EA com a condução de outro estudo de caso, o que permite um melhor desempenho na execução desse estudo e a validade de construção que descreve que estudos de casos mais complexos são necessários para avaliar o processo e a utilização do processo por uma empresa desenvolvedora de *software* pode contribuir para sua maturidade.

4.5 Operação

Execução: O estudo de caso foi conduzido em duas etapas. A primeira etapa teve como objetivo identificar o domínio do projeto de *software* em relação ao *framework* disponível e avaliar a viabilidade em realizar o projeto. Para a realização desta etapa foram executadas as duas primeiras atividades da fase de CONCEPÇÃO do processo, sendo: "Familiarizar com o domínio do framework" e "Observar o domínio do projeto de software em relação ao framework". Para a execução destas atividades foram gastas 72 horas, devido ao projeto apresentar uma particularidade diferente a projetos anteriores, como o controle de dois recursos, produtos e serviços, o que exigiu aprendizado.

A segunda etapa consistiu na confirmação do início do projeto de *software* e na aplicação por completo do PARFAIT/EA no desenvolvimento do *software*. Durante esta etapa foi registrado o tempo gasto na execução de cada atividade e em cada ciclo de desenvolvimento, como apresentado na Tabela 1.

Para este estudo de caso não foi utilizada nenhuma ferramenta de teste para apoiar a atividade de teste da fase de CONSTRUÇÃO. A técnica aplicada foi a técnica de Teste Funcional e os critérios foram "Particionamento em Classes de Equivalência" e "Análise de Valor Limite" [13]. O Teste de Aceitação [13] foi também constantemente aplicado durante o estudo de caso, a fim de avaliar o *software* produzido juntamente com o cliente. Na atividade de teste da fase de TRANSIÇÃO do processo, apenas o Teste de Aceitação foi aplicado.

De acordo com o planejamento do projeto definido na Seção 4.1, três ciclos de desenvolvimento foram necessários. No primeiro ciclo foram disponibilizados os módulos do *software* de cadastro e seus respectivos relatórios. Já, no segundo ciclo foram disponibilizados

os módulos principais do *software*, sendo: o módulo de "Empréstimo de Revistas" e o de "Lançamento de Serviços", juntamente com os relatórios gerenciais. No terceiro ciclo foram realizadas apenas adaptações nas telas para uma melhor apresentação do *software.*

Na Tabela 1 apresentam-se as atividades executadas em cada ciclo de desenvolvimento, bem como o tempo gasto em cada atividade. Já na Tabela 2 apresentam-se os dados coletados referentes às variáveis dependentes, definidas na Seção 4.4.

4.6 Análise e Interpretação dos Resultados

A condução do estudo de caso não foi realizada com horários contínuos e todo o tempo gasto nas atividades em cada ciclo de desenvolvimento foi anotado.

Observou-se que a execução da primeira atividade da fase de CONCEPÇÃO, "Familiarizar com o domínio do framework" **ajuda** não somente no entendimento do *framework,* como também permite analisar o uso dos padrões da LPA em projetos semelhantes, contribuindo para o entendimento do novo projeto.

Verificou-se que a maior parte do tempo gasto foi na atividade "Desenvolver o diagrama de casos de uso e elaborar os casos de teste", principalmente na elaboração dos casos de teste, como constatado por Cagnin [9]. O uso de *framework* no projeto de *software* permite **reusar** diagramas de casos de uso de projetos anteriores semelhantes ao projeto atual, devido às funcionalidades semelhantes produzidas do domínio do *framework*, diminuindo o tempo para a descrição da funcionalidade. Na descrição dos casos de uso, foram gastas 3h10 e na elaboração dos casos de teste 34h20, o que corresponde a um total de 37h20 gastos nessa atividade em todo o projeto.

Pelo fato do projeto de *software* ser simples e possuir a maioria dos requisitos do *framework*, algumas atividades foram executadas em pouco tempo, como exemplo a atividade "Documentar as modificações realizadas no diagrama de classes", em que foi gasto 00h30 para a descrição dos atributos adicionados no diagrama de classe e a descrição do padrão que não atendeu completamente a regra de negócio do *software*. Tal regra determina que o prazo de empréstimo de uma revista é de três dias e o atraso na devolução da revista é de dois dias de suspensão para cada dia de atraso. Essa regra de negócio foi identificada somente no segundo ciclo após sua implantação. No início do projeto essa regra de negócio não existia.

Neste estudo de caso não foi marcado o tempo gasto com a aplicação das inspeções nas atividades e o uso dos marcos de referência foi associado à atividade "Reuniões freqüentes", executada no final de cada atividade.

Com a totalização do tempo gasto, verificou-se que no primeiro ciclo foi gasto o maior tempo do projeto, o que caracteriza o processo **"dirigido ao risco"**, pelo fato desse ciclo conter o núcleo do projeto de *software*, o qual define uma arquitetura sólida com o apoio do *framework*.

No terceiro ciclo, foram realizados apenas alguns ajustes nas telas do *software,* desabilitando funcionalidades presentes no *framework* e ausentes no projeto. Nesse ciclo foi gasto um total de 04h20.

Um ponto importante do PARFAIT/EA definido neste estudo de caso é a utilização de **diretrizes** e **inspeções** aplicadas em algumas atividades do processo. O objetivo das diretrizes é apoiar a execução do passo correspondente e as inspeções de avaliar os artefatos produzidos, como exemplo a atividade "Criar o software", da fase de CONSTRUÇÃO, que possui uma inspeção com o objetivo de verificar se todos os padrões da LPA, utilizados para criar o diagrama de classes e os atributos adicionais, foram considerados na instanciação do *framework*. Algumas diretrizes (D) e inspeções (I) apóiam apenas um único passo na atividade e outras apóiam toda a atividade.

Com a aplicação do estudo de caso e o resultado apresentado em relação às práticas ágeis, observou-se a falta da definição de práticas ágeis existentes na literatura em algumas atividades, como na atividade "Adaptar o software". Após a condução do estudo de caso, definiu para a essa atividade as práticas do método XP (BECK, 2000): projetar com simplicidade e propriedade coletiva do código, em que a primeira atividade foi modificada para "Projetar (adaptar) com simplicidade".

Pelo fato do estudo de caso ser um estudo observacional [19], não foi possível obter dados conclusivos com relação às hipóteses formuladas. No entanto, foi possível observar os resultados obtidos, em que houve uma grande motivação em conduzir novos estudos de casos em projetos maiores.

Tabela 1 – Dados coletados durante cada ciclo de desenvolvimento

Fase	Atividade	Ciclo 1	Ciclo 2	Ciclo 3
1	Observar o domínio do projeto de software em relação ao *framework*	72h00	01h00	-
1	Elaborar o planejamento do projeto de software	00h20	00h30	00h20
1	Confrontar as características não funcionais do *framework* x projeto de software	00h20	00h20	-
1	Reuniões freqüentes	00h15	00h15	-
2	Desenvolver o diagrama de casos de uso e elaborar os casos de teste	19h30	18h00	-
2	Documentar as regras de negócio do *software*	-	00h06	-
2	Desenvolver o diagrama de classes do *software*	00h30	00h20	-
2	Documentar as modificações realizadas no diagrama de classes	2h30	00h20	-
	Reuniões freqüentes	00h15	00h15	-
3	Criar o *software*	00h20	01h06	-
3	Executar os casos de teste no *software*	03h00	03h00	-
3	Adaptar o *software*	01h00	04h00	02h30
3	Reuniões freqüentes	00h15	00h15	00h15
4	Testar o *software*	01h00	01h00	01h00
4	Treinar os usuários finais	00h20	00h30	00h15
4	Reuniões freqüentes	00h15	00h30	00h15
	Total Geral do ciclo	**100h55**	**31h12**	**04h20**

Tabela 2 - Dados coletados durante o estudo de caso

Dado coletado nas hipóteses nulas
Lista de padrões de análise utilizados, diagrama de classes e lista de requisitos foram suficiente para instanciar o framework em cada iteração do processo? **Resultado:** Sim, suficiente.
Lista de padrões de análise utilizados, diagrama de classes e lista de requisitos foram suficientes para as adaptações realizadas no software em cada iteração do processo? **Resultado:** Sim, tais artefatos foram suficientes para adaptações realizadas no desenvolvimento. No entanto são necessários novos estudos para as adaptações (manutenção) após a entrega do *software.*
Quantidade de regras de negócio identificada em cada ciclo de desenvolvimento **Resultado:** Apenas uma regra de negócio foi identificada no segundo ciclo, devido às mudanças ocorridas na empresa.
Quantidade e identificação de práticas ágeis utilizadas em cada iteração do processo? **Resultado:** No primeiro ciclo foram utilizadas 100% das práticas ágeis definidas no processo, sendo: cliente presente, metáforas, jogo do planejamento, detalhamento dos requisitos definidos, versões freqüentes, integração contínua e testes constantes, já no segundo ciclo foram utilizadas 50 % das práticas definidas, sendo: cliente presente, jogo do planejamento, detalhamento dos requisitos definidos, integração continua e testes constantes e no terceiro ciclo foram utilizadas 30 % das práticas definidas, sendo: cliente presente, jogo do planejamento e testes constantes.
Quantidade de regras de negócio identificada na definição da lista de requisitos? **Resultado**: Nenhuma regra identificada.

De acordo com os resultados apresentados na Tabela 2, a diminuição do uso das práticas ágeis do ciclo um para o ciclo dois, é devido ao fato de que algumas práticas ágeis, como por exemplo, o uso de metáforas já estarem definidas no projeto. Já a diminuição das práticas ágeis do ciclo dois para o três, baseia-se no contexto do ciclo, em que foram realizadas apenas adaptações nas telas.

Em relação a não identificação de regras de negócios na lista de requisitos, pode ser justificada pelo fato do projeto ser simples. Vale ressaltar que a regra de negócio descrita no estudo de caso foi definida

durante o andamento do projeto, após uma reunião com os funcionários da empresa incubadora.

5. Evolução do PARFAIT/EA

Na versão do PARFAIT/EA foi adicionado a cada atividade do processo o profissional responsável (papel) pela execução da atividade, os artefatos de entrada utilizados para executar a atividade e os artefatos produzidos em cada atividade.

A escolha dos papéis em cada atividade foi baseada nos papéis indicados pelo processo PARFAIT [7] e em um estudo realizado sobre alguns métodos ágeis, como: XP (*Extreme Programming)* [4]*, Scrum* [1]*,* FDD (*Feature-Driven Development*) [1], ASD (*Adaptative Software Development*) [1], DSDM (*Dynamic Systems Development Method*) [1], uma vez que cada método apresenta o responsável (papel) para a execução de cada atividade.

Um exemplo de uma atividade do processo PARFATIE/EA que foi evoluída é a atividade "Executar os casos de teste no *software*", classificada como atividade obrigatória. Os papéis sugeridos para a execução dessa atividade são: o testador e a participação do cliente. A prática ágil aplicada nessa atividade são testes constantes e os artefatos de entrada que devem existir para que a atividade possa ser executada são: documentação dos casos de teste e o *software*. O artefato de saída que deve ser produzido é o relatório resumo de teste. Essa atividade possui ainda uma diretriz que deve ser aplicada no passo dois, ou seja. "Comparar o resultado obtido do software com o resultado esperado de cada caso de teste" e uma outra diretriz aplicada no passo três "Validar juntamente com o cliente as regras de negócio e os requisitos identificados". Essa atividade possui uma única inspeção que é aplicada em toda atividade, ou seja, "Garantir que todos os casos de testes documentados sejam executados".

Verificou-se que a utilização de diretrizes e inspeções no processo contribuiu para sua evolução e para a qualidade do *software* produzido.

Com a evolução do processo PARFAIT/EA a atividade "Converter a base de dados do software" da fase de TRANSIÇÃO foi retirada, seguindo o contexto em que o cliente não possui dados para conversão.

6. Conclusões e Trabalhos Futuros

Com a condução do estudo de caso, observou-se que a aplicação de inspeções em algumas atividades do processo apóia a Verificação e Validação (V&V) e, conseqüentemente, aumenta a qualidade do *software* desenvolvido.

Outros estudos de casos serão necessários, com objetivo de avaliar alguns pontos do processo, como exemplo o uso de ferramenta de teste para a execução dos casos de testes e ferramentas para controlar as versões de *software*

O ideal para o próximo estudo de caso é que um outro participante aplique o processo no desenvolvimento de *software*, obtendo-se outros resultados.

Como trabalhos futuros, têm-se: a construção de uma ferramenta para documentar projetos de desenvolvimento de *software* que utilizem um *framework* baseado em uma LPA e planejar um estudo de caso para aplicar as tarefas de manutenção de *software*, sendo: de correção, de adaptação e de aperfeiçoamento, com o objetivo de avaliar se a documentação produzida pelo processo PARFAIT/EA é suficiente e se as atividades do processo apóiam essa fase.

7. Referências

[1] ABRAHAMSSON, P.; SALO, O.; RONKAINEN, J.; WARSTA, J. **Agile Software Development Methods**: **Review and Analysis.** ESPOO (Technical Research Centre of Finland), 2002.

[2] AMBLER , S. W. **Modelagem Ágil.** Editora: Bookman, 2004.

[3] APPLETON, B. **Patterns and software: Essential concepts and terminology.** URL: http://www.cmcrossroads.com/bradapp/docs/patterns-intro.html, 1997.

[4] BECK, K. ***Extreme Programming explained: Embrace change***. Second ed. Addison-Wesley, 2000.

[5] BRAGA, R. T. V. **Um Processo para Construção e Instanciação de Frameworks baseados em uma Linguagem de Padrões para um Domínio Específico**. Tese de Doutoramento, ICMC-USP, São Carlos-SP, 2003.

[6] BRAGA, R. T. V.; MASIERO, P. C. **Building a wizard for framework instantiation based on a pattern language**. In: *OOIS'2003, International Conference on Object-Oriented Information Systems, 9*, Geneva, Switzerland: Lecture Notes on Computer Science, LNCS 2817, Springer, p. 95-106, 2003.

[7] CAGNIN, M. I. **PARFAIT: Uma Contribuição para Reengenharia de Software baseada em Linguagens de Padrões e Frameworks.** Tese de Doutoramento, ICMC-USP, São Carlos-SP, 2005.

[8] CAGNIN, M. I.; MALDONADO, J. C.; BRAGA, R. T. V.; GERMANO, F. **GREN-WizardVersionControl: Uma Ferramenta de Apoio ao Controle de Versão das Aplicações Criadas pelo Framework GREN**. In: Sessão de Ferramentas'2004, Simpósio Brasileiro de Engenharia de Software, Brasília, DF, p. 73-78, 2004.

[9] CAGNIN, M. I.; MALDONADO, J. C.; CHAN, A.; PENTEADO, R. D.; GERMANO, F. S. **Reuso na Atividade de Teste para Reduzir Custo e Esforço de VV&T no Desenvolvimento e na Reengenharia de Software**. In: *XVIII* Simpósio Brasileiro de Engenharia de Software, Brasília, DF, p. 71-85, 2004.

[10] DANTA, V., GARCIA, F.P., LIMA. ***easYProcess*: Um Processo De Desenvolvimento De Software para Uso No Ambiente Acadêmico.** XII WEI - Workshop de Educação em Computação, Salvador, BA, 2004.

[11] FAYAD, M. E.; JOHNSON, R. E. **Domain-specific Application Frameworks: Frameworks Experience by Industry**. First ed. John Wiley & Sons, 2000.

[12] GOMES, F. D.; CAGNIN, M. I.; MALDONADO**,** J. C. **Esboço de um Processo Ágil de Desenvolvimento baseado em Framework**. In: CLEI'2006, 32th Conferência Latino Americano de Informática, Santiago – Chile, 2006.

[13] MYERS, G. J. **The art of software testing**. second ed. Wiley, 2004.

[14] PFLEEGER, S. L. **Engenharia de software: teoria e prática**. 2ª ed. São Paulo: Prentice Hall, 2004.

[15] PRESSMAN, R.S. **Engenharia de Software**. 5ª ed. Rio de Janeiro: McGraw-Hill, 2002.

[16] KRUCHTEN, P. ***The Rational Unified Process**: An Introduction*. Second ed. Addison-Wesley, 2000.

[17] SAMPAIO, A., VASCONCELOS, A., SAMPAIO, P. R. F. **Design and Empirical Evaluation of an Agile Web Engineering Process.** XVIII Simpósio Brasileiro de Engenharia de Software, Brasília – DF, Brasil, p. 194-209, 2004.

[18] SOMMERVILLE, I. **Engenharia de Software.** Editora: Addison Wesley, 2003.

[19] WHOLIN, C.; RUNESON, P.; HÖST, M.; OHLSSON, M.; REGNELL, B.; WESSLÉN, A. **Experimentation in software engineering**: An introduction. Kluwer, 2000.

Desarrollo de un código de métricas para pequeñas empresas ecuatorianas desarrolladoras de software

Raúl González Carrión.
Ingeniero en Computación, Escuela Superior Politécnica del Litoral.
Facultad de Ingeniería en Electricidad y Computación.
e-mail:rgonzale@espol.edu.ec

Henry Hernández Rendón.
Ingeniero en Computación, Escuela Superior Politécnica del Litoral.
Facultad de Ingeniería en Electricidad y Computación.
e-mail: hhernande@fiec.espol.edu.ec

Mónica Villavicencio Cabezas.
Ingeniera en Computación, Escuela Superior Politécnica del Litoral.
Facultad de Ingeniería en Electricidad y Computación.
e-mail: mvillavi@espol.edu.ec

Resumen

El presente artículo reporta un estudio orientado a validar el instrumento de medición que se utilizó para la obtención de métricas. El estudió se orientó también a determinar los factores que inciden en la administración de proyectos de software en empresas pequeñas desarrolladoras de software en el Ecuador. Se tomó una muestra de 15 empresas asentadas en las ciudades de Quito y Guayaquil. Los resultados que se presentan tienen relación con la variación del porcentaje de error en la estimación del esfuerzo, la determinación del porcentaje de esfuerzo empleado en documentación; la complejidad técnica y del negocio en los proyectos; el establecimiento del porcentaje de error en la estimación del costo; el conocimiento del costo promedio hora; la presencia de cambios en los requerimientos y su frecuencia; la evaluación de defectos y fallas por fases y su severidad; la eficiencia de las empresas para atender los defectos y las fallas presentados, entre otros. El estudio pretende cuantificar la eficacia en la planificación, la eficiencia en la administración de recursos, la frecuencia y severidad de los elementos conspiradores y el grado de efectividad con que se superan tales elementos en empresas pequeñas.

1. Introducción

Dos estudios realizados por el Sub-Componente 8 de Ingeniería de Software [1] [2], dentro del marco de Proyecto VLIR-ESPOL, sirvieron como punto de partida para la elaboración de un plan de métricas, así como los instrumentos utilizados para la recopilación de datos en el contexto ecuatoriano [3]. Previamente, se realizó un plan piloto en el que se aplicó un instrumento de medición en tres empresas grandes desarrolladoras de software ubicadas en las ciudades de Guayaquil y Quito [4]. Seguidamente, se dio inicio al presente estudio haciendo un corte a la información generada por el estudio piloto, contándose también con el plan de métricas con las respectivas plantillas para el levantamiento de información. Esto último facilitó redefinir y adicionar parámetros y, consecuentemente, formular los indicadores definitivos para la presente investigación [5].

2. Población objetivo del estudio

Para la elección de la población objetivo, se consideró como elemento fundamental la premisa técnica de que, para realizar una correcta comparación de los datos, era necesario que las empresas tuvieran similares características. En tal sentido, se han considerado los siguientes principios:

Seleccionar únicamente empresas que dentro de sus líneas de negocios tengan el desarrollo de software y/o la provisión de servicios asociados. Por lo tanto, se excluyó las empresas que tienen como actividad exclusiva la comercialización.

Que las empresas sean de tamaño pequeño, tomando como base el número de empleados. Para esto, se hizo uso de estudios previos dentro del Proyecto VLIR-ESPOL, Componente 8 Área de Ingeniería de Software, realizados en los años 2003 [1] y 2005 [2], donde se aplicaron los siguientes rangos:

- Pequeñas: 10 empleados o menos.
- Medianas: entre 10 y 50 empleados.
- Grandes: 50 empleados o más.

Utilizando estas definiciones, se procedió a seleccionar las empresas pequeñas para el presente estudio.

Que los proyectos sean de tamaño pequeño, con base en el criterio de la duración de éstos. Para lo cual, se utilizó como término de referencia los mismos estudios realizados dentro del Proyecto VLIR-ESPOL en los años 2003 [1] y 2005 [2], en los que se clasifica a los proyectos pequeños a aquellos cuya duración está entre 1 y 6 meses; proyectos medianos a los que están entre 7 y 15 meses; y grandes a los que requieren más de 15 meses para su ejecución [1].

Que su proceso de desarrollo de software se divide en fases: planificación, especificación, diseño, construcción, pruebas e instalación. Dichas fases pueden ser ejecutadas en cualquier orden, no necesariamente se debe haber terminado una para dar inicio a otra.

3. Muestra seleccionada

Para la selección de la muestra, se partió de un universo de empresas tomado de la base de datos del Componente 8 del Proyecto VLIR – ESPOL, la cual agrupa un total de 200 empresas, ubicadas en las ciudades de Quito, Guayaquil y Cuenca. Esta base de datos contiene información actualizada de todas las empresas que desarrollan software en el Ecuador. De este número de empresas, se tomó una muestra de 20 empresas desarrolladoras de software, en mayo del 2006; decisión que responde al principio estadístico de que cuando se realizan estas pruebas, es suficiente tomar el 10% del universo de estudio [6].

Con la finalidad de cumplir con el tamaño de muestra de 20 empresas, fue necesario visitar y/o llamar a un total de 73 empresas que tuvieran el perfil de la población objetivo del estudio. Al término de esta actividad, se obtuvo que: 13 empresas no aceptaron participar (rechazo explícito); 1 no desarrolla software; 7 no tenían proyecto para aplicar las métricas; 7 estaban participando en otros proyectos de investigación; 4 quedaron pendientes de confirmar (nunca confirmaron); 3 manifestaron interés, pero al final nunca participaron; 18 nunca se pudo localizar a la persona encargada; y, finalmente, 20 empresas aceptaron la participación. A estas últimas se les proporcionó el programa de instrucciones. De estas 20 empresas, 5 no cumplieron el compromiso adquirido, por lo que se optó por solicitar a las 15 restantes que aporten con proyectos adicionales para completar al menos 20 proyectos. [5]

4. Instrumento de medición

Se define como instrumento de medición a la herramienta que sirvió para lograr el objetivo fundamental del presente estudio; obtener métricas de empresas desarrolladoras de software. Su mecanismo de funcionamiento se apoya en el levantamiento de información, cálculo, y evaluación de los resultados obtenidos a fin de inferir acciones y recomendaciones tendientes a optimizar los resultados en esta parte de la industria informática [5]. De esto, se derivan los indicadores que se explican a continuación. En este trabajo no han sido incluidos todos los resultados del estudio, pues a la fecha de presentación del artículo no se había concluido con todo el análisis.

5. Resultados de los indicadores obtenidos

5.1. Resultados indicador 1: porcentaje de error en la estimación del esfuerzo

Este indicador determina el grado de acercamiento de los proyectos a sus estimaciones de tiempo, con lo cual se obtuvo que:

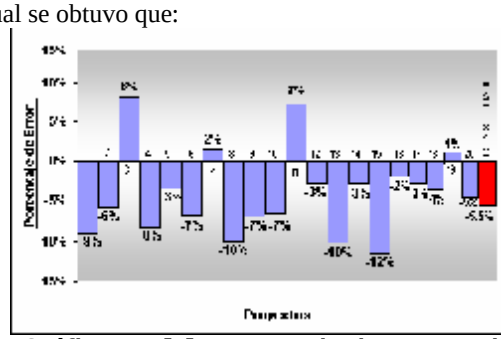


Gráfico 5.1 [5]. Porcentaje de error en la estimación del esfuerzo

El promedio general de todos los proyectos estimó el 5.5% menos del tiempo necesario para desarrollarlos, lo que explica la posición negativa que se evidencia en las barras.

La gráfica evidencia que en dos de los proyectos, esto es, el 7 y el 19, el nivel de estimación de tiempo es

apropiado, pues, a pesar de haberse estimado más del tiempo requerido, dicho exceso únicamente se ubica en el 2% y 1% respectivamente. Estas discrepancias se consideran razonables en estos casos.

El sesgo en el proyecto 15 indica que se estimó un 12% menos del tiempo total requerido, situación que ratifica la tendencia en el sentido de que la gran mayoría de los proyectos informáticos carecen de una adecuada estimación. [5]

5.3. Resultados indicador 2: porcentaje del esfuerzo empleado en documentación

Este indicador refleja el porcentaje de las horas totales que cada proyecto ha dedicado a su documentación. [5]

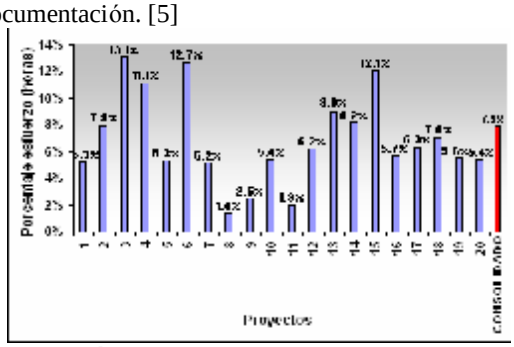


Gráfico 5.2. Porcentaje del esfuerzo empleado en documentación [5]

El consolidado de todos los proyectos participantes alcanza el 7.9%, valor que en la práctica se considera relativamente bajo debido a que la experiencia de personas conocedoras de estos temas sugieren mayor dedicación de horas al proceso de documentar los proyectos, por el alto riesgo que su ausencia implica[10]. Se evidencia en los proyectos 3 un 13.1% y en el 6 un 12.7%, en el 15 un 12.1% y en el 4 un 11.1%, los mismos que por exceder de manera evidente al promedio, se consideran datos cuya atipicidad estaría dada porque se trata de nuevos proyectos o porque el encargado de esta función no posee la experiencia necesaria en el manejo de técnicas de documentación, y no porque posean cultura de documentación. [5]

5.3. Resultados indicador 3: complejidad del negocio y complejidad técnica

Este indicador relaciona la complejidad del negocio con la complejidad técnica, lo que apunta a una separación de los proyectos participantes en 4 cuadrantes, de tal manera que se evidencia la virtual segregación existente entre proyectos simples, muy complejos, técnicamente complejos y negocios de complejidad implícita. [5]

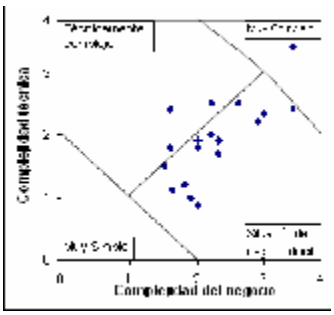


Gráfico 5.3. Complejidad de los proyectos [5]

De la grafica se puede deducir que la mayor parte de proyectos han sido realizados para negocios complejos; al tiempo que se aprecia pocos proyectos que han derivado alta complejidad. Adicionalmente encontramos que en el rango de proyectos muy complejos existiría un solo caso. La gráfica muestra también de manera clara que no han existido proyectos que puedan catalogarse en la calidad de muy simples.[5]

5.4. Resultados indicador 4: porcentaje de error en la estimación del costo

Este indicador determina el grado de acercamiento de los proyectos a sus estimaciones presupuestales de costo. [5]

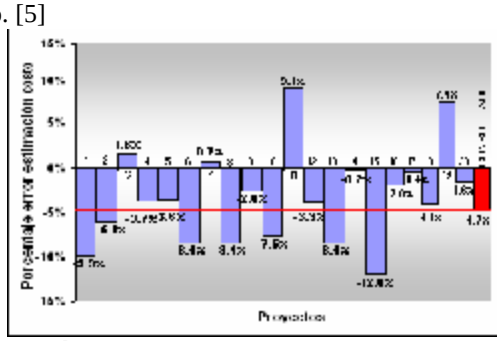


Gráfico 5.4. Porcentaje de error en la estimación del costo [5]

El promedio general de todos los proyectos estimó el 4.7% menos del costo necesario para desarrollarlos, lo que explica la posición negativa que se evidencia en las barras.

Los proyectos 7, 14 y 17 acusan un apropiado nivel de estimación de costo, considerando que, a pesar de haber estimado más o menos el costo requerido, dicho exceso o faltante, según el caso, únicamente se discrepa en el 0.7%, -0.2% y -0.4% respectivamente.

Varios proyectos se pasaron de la media, situación que ratifica la tendencia en el sentido que la gran mayoría de los proyectos informáticos carecen de una adecuada estimación en el costo, lo que respondería al escaso esfuerzo en la planificación los mismos. [5]

5.5. Resultados indicador 5: costo promedio hora real

Este indicador refleja el costo promedio hora real de los proyectos participantes. [5]

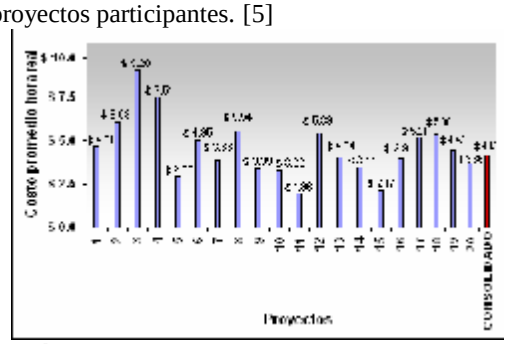


Gráfico 5.5. Costo promedio hora real [5]

El consolidado de todos los proyectos alcanza un costo promedio de $4.12, que si se lo examina con los demás participantes, observamos que 10 proyectos están por arriba y 10 por debajo de dicho promedio, lo que nos llevaría a concluir el por qué de la tendencia.

El proyecto 3 tiene un costo de $9.20 promedio hora real, valor que por exceder de manera evidente a la media se considera atípico, en razón de que en el mismo habrían participado en mayor proporción recursos de más alto costo en la empresa, como el caso de gerentes o líderes de proyectos. Podría también haber la presencia de costos indirectos adicionales que inciden en el promedio hora real, como el caso de personal indirecto que habría participado en los proyectos, tales como secretarias, personal de apoyo y hasta eventuales honorarios a consultores.

Por otro lado, el proyecto 11 tiene un costo hora promedio de $1.96, cuya atipicidad podría obedecer a que en el proyecto participaron recursos de bajo costo, los que no necesariamente podrían catalogarse como malos recursos.

Como dato adicional tenemos un artículo publicado por la CORPEI en el que indican los resultados de la encuesta salarial realizada por Price Waterhouse SIREM "Servicio Integrado de Remuneraciones" a Empresas Nacionales y Multinacionales del Ecuador [11]. Dicho artículo presenta lo siguiente:

Costo hora promedio:

Ingeniero / Analista de Sistemas Senior = $6.18 (calculado de 160 horas).

Ingeniero / Analista de Sistemas Junior = $3.80 (calculado de 160 horas).

5.6. Resultados indicador 6: porcentaje de inspecciones realizadas por fases

Este indicador determina a nivel consolidado la proporción que representa las inspecciones realizadas en cada fase, respecto al total de tareas realizadas en las mismas. [5]

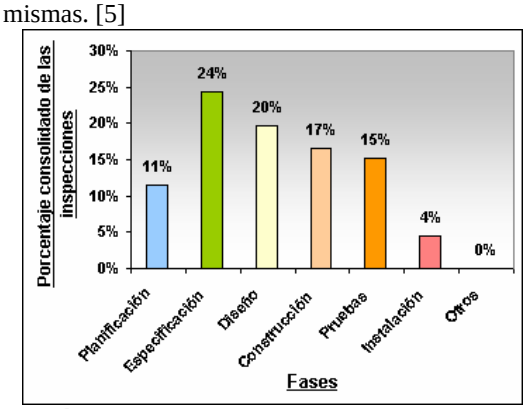


Gráfico 5.6. Porcentaje de inspecciones realizadas por fases [5]

La mayor cantidad de inspecciones se realizan a partir de la segunda fase, esto es, en la especificación, donde se observa que las fases siguientes tuvieron niveles de inspección porcentualmente decrecientes. A nuestro parecer las empresas deben de inspeccionar también su plan de trabajo para prever de manera temprana complicaciones en el desarrollo del proyecto.

5.7. Resultados indicador 7: porcentaje de variación de los requerimientos

Este indicador mide la forma como han variado en porcentaje los requerimientos originalmente propuestos respecto a los ejecutados en el proyecto. [5]

Previo a la discusión acerca de los resultados obtenidos del presente indicador, es necesario explicar que significa "requerimiento" para las empresas. Un requerimiento "es una funcionalidad que actualmente no posee el sistema de software o cualquier funcionalidad que se le pueda agregar de forma o de fondo".

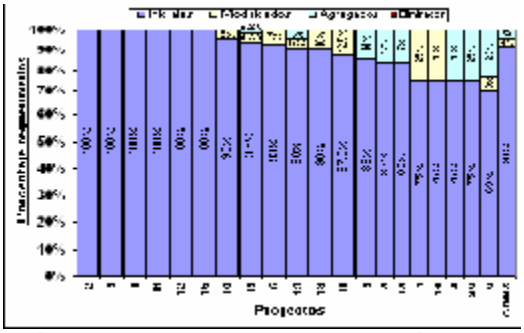


Gráfico 5.7. Porcentaje de variación de los requerimientos [5]

El promedio general consolidado respecto a los requerimientos iniciales en todos los proyectos

participantes alcanzó el 91%, esto es, que de los requerimientos iniciales el 91% fueron ejecutados, el 4% fueron modificados y un 5% fueron nuevas solicitudes para el sistema.

Existen 11 proyectos en los que se evidencia un porcentaje de requerimientos iniciales ejecutados, inferior al promedio general de 91%, entre los que se encuentran los proyectos 1, 14, 4, 20, con un porcentaje de 75% de requerimientos iniciales ejecutados y el proyecto 8 con 69% de requerimientos iniciales ejecutados.

Respecto al porcentaje promedio general consolidado, el 4% que representa los requerimientos modificados, encontramos que 9 proyectos se encuentran por arriba de dicho porcentaje, aspecto que reflejaría que éstos realizaron un deficiente levantamiento de información o que podría existir falta de experiencia o conocimiento del entorno del cliente y/o usuario.

Del 5% que representan los requerimientos agregados, se determinó que 7 proyectos superan dicho promedio y uno está por debajo, aspecto que en ambos casos reflejaría una inadecuada coordinación con los clientes y/o usuarios.

Respecto a los requerimientos eliminados, en la gráfica se puede observar que, de la muestra seleccionada, no se ha presentado dicho particular. [5]

5.8. Resultados indicador 8: ocurrencia de defectos y fallas por fases

Este indicador mide la proporción que representa la ocurrencia de defectos y/o las fallas con respecto a la totalidad de las tareas a nivel consolidado. [5]

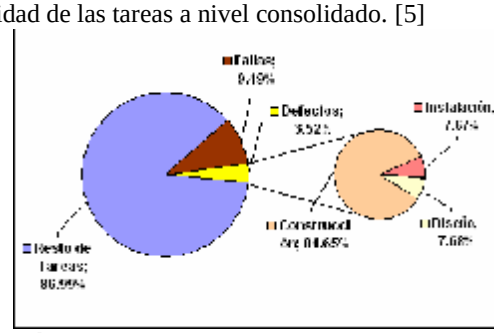


Gráfico 5.8.1. Ocurrencia de defectos por fase [5]

Con relación a la ocurrencia de defectos, se encontró que el 3.52% de las tareas se orientó a atender los defectos presentados, de los cuales se establece que el 84.65% se concentran en la fase de construcción, el 7.68% en la fase de instalación y el 7.67% de estos en la fase de diseño. [5]

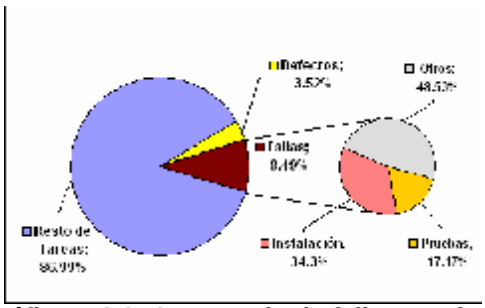


Gráfico 5.8.2. Ocurrencia de fallas por fase [5]

Con relación a la ocurrencia de fallas, se evidenció que el 9.49% de las tareas se orientó a atender las fallas presentadas, de las cuales se establece que el 48.53% se concentran en la fase "otros", también llamada fase de mantenimiento o post implementación, el 34.3% en la fase de instalación y el 17.17% de estos en la fase de pruebas. [5]

5.9. Resultados indicador 9: tipo de defectos y fallas más frecuentes (identificados por severidad)

Este indicador mide la proporción que representa los tipos de defectos y/o fallas con respecto a la totalidad de las tareas, identificados por su severidad [5]

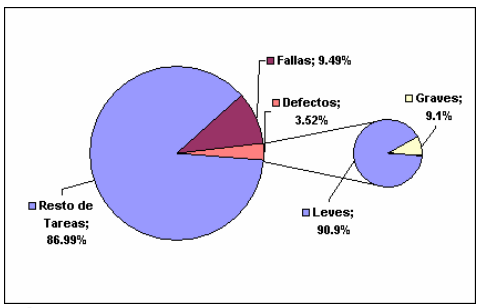


Gráfico 5.9.1. Porcentaje consolidado de defectos respecto a las tareas (identificadas por su severidad) [5]

La presente gráfica corresponde a la proporción de los defectos con respecto al total de tareas, la misma que alcanza el 3.52%. De estos defectos, el 90.9% fueron de carácter leve, el 9.1% graves y 0% críticos. [5]

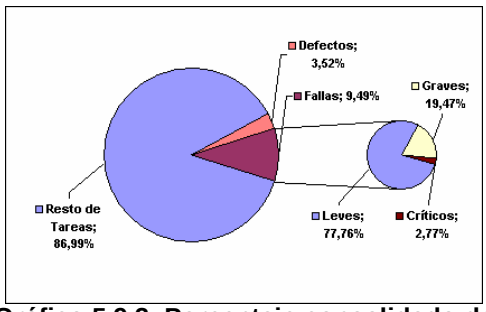


Gráfico 5.9.2. Porcentaje consolidado de fallas respecto a las tareas (identificadas por su severidad) [5]

La presente gráfica, que también se analizó en el indicador anterior corresponde a la proporción de las fallas que, con respecto al total de tareas, alcanzan el 9.49% lo que indica que por su severidad, el 77.76% fueron de carácter leve, el 19.47% graves y el 2.77% críticos. [5]

5.10. Resultados indicador 10: eficiencia en atención de defectos y fallas

Este indicador revela la eficiencia de la empresa proveedora de software en corregir los defectos y/o fallas. La eficiencia fue medida en número de días, es decir los días que transcurrieron desde la fecha de notificación del defecto y/o falla hasta la fecha de reparación de los mismos. Este indicador es conocido también como tiempo de respuesta [9] o promedio de ciclo [5].

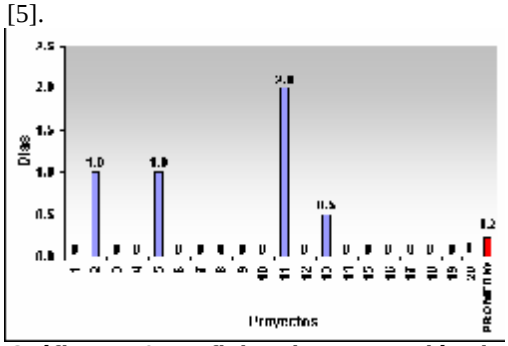


Gráfico 5.10.1. Eficiencia en atención de defectos [5]

En la gráfica de arriba, el consolidado de todos los proyectos evidencia un tiempo de respuesta promedio de 0.2 días, esto es, aproximadamente 5 horas.

En 16 proyectos no se registró este indicador debido, fundamentalmente, a que las empresas no tienen como práctica hacerlo y no cuantifican ni califican los defectos. Los defectos son reportados por terceros como fallas. [5]

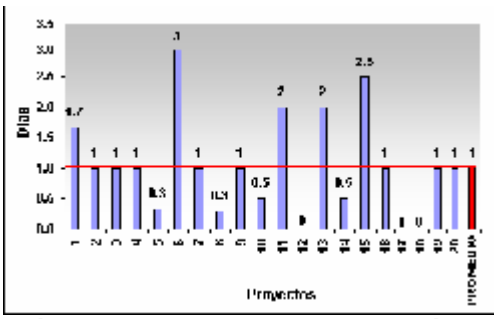


Gráfico 5.10.2. Eficiencia en atención de fallas [5]

En esta gráfica, el consolidado de todos los proyectos evidencia un tiempo de respuesta promedio de 1 día para atender las fallas reportadas. Podemos observar en la gráfica 5,10.2 que 5 proyectos están por arriba del promedio y 7 por debajo. Ocho proyectos son iguales al promedio, lo que nos lleva a concluir que existe una tendencia marcada. [5]

6. Conclusiones

Es importante dedicar una considerable cantidad de tiempo a la documentación de los procesos, puesto que el requisito fundamental para obtener la certificaciones ISO9001:2000 ò CMMI es que la empresa tenga y lleve los procesos debida y correctamente documentados [7] [8].

Respecto a la complejidad de los proyectos, se evidenció una alta presencia de proyectos de negocios complejos, con pocos proyectos de complejidad técnica y solamente un proyecto catalogado como complejo. No se tuvo proyectos de complejidad simple, aspecto que ratifica la buena selección de la muestra.

Respecto a la estimación del costo, 16 proyectos han estimado por encima o por debajo de lo presupuestado, específicamente la gran mayoría estimó menos de lo que realmente gastó.

La realidad del mercado salarial ha evidenciado que el costo hora de desarrollar software en Ecuador es relativamente bajo, pese a la calidad de profesionales que trabajan desarrollando proyectos. Esto se da muy particularmente en empresas pequeñas. [5]

En estudios previos se logró determinar que el costo promedio hora varía según la categoría de recursos que participan en los proyectos, de tal manera que los proyectos en los que intervienen categorías jerárquicas altas, producen mayor costo hora promedio y, contrariamente, los proyectos en los que intervienen categorías jerárquicas de menor nivel, tienen bajo costo promedio hora. En el presente estudio, se evidenció un relativo equilibrio en este sentido. [5]

Se confirma la poca realización de inspecciones en la fase de planificación, aspecto que lleva a las

empresas a incurrir en errores de costo y estimación de los proyectos.

Indicadores como la variación porcentual en los requerimientos proporcionan información que se considera de alta utilidad práctica para futuros proyectos, pues los encargados de ejecutarlos pueden analizar y planificar apropiadamente el levantamiento de información y la coordinación con los clientes y/o usuarios. [5]

La mayor proporción de los defectos y fallas presentados en el estudio, catalogados por su severidad, se ubican en la categoría de leves, aspecto que no entraña mayor riesgo debido a que la demanda de recursos para su regularización es relativamente baja. [5]

Los resultados obtenidos respecto al porcentaje de defectos son considerablemente bajos, lo que indicaría ausencia de documentación cuando estos se presentan. Lo que a su vez confirmaría la aseveración de algunas empresas en el sentido de que, "para los programadores su software es casi perfecto, que nunca tiene defectos y sólo se los encuentra cuando el líder del proyecto o el cliente/usuario se da cuenta de aquello. Todo esto siempre en la fase de pruebas", opinión que no es compartida por los autores de este estudio debido a que no se están documentando apropiadamente los defectos cometidos en las fases anteriores. [5]

Para realizar este tipo de estudios, es necesario, además de un concienzudo análisis, una prolija investigación junto a una actitud perseverante y persistente, pues las empresas ecuatorianas si bien están interesadas en este tipo de proyectos, desafortunadamente no disponen de tiempo ni recursos para dar el apoyo requerido [5]

10. Referencias

[1] Danny Salazar, Mónica Villavicencio, María Macías, Monique Snoeck. "Estudio estadístico exploratorio de las empresas desarrolladoras de software asentadas en Guayaquil, Quito y Cuenca", ESPOL – VLIR, Componente 8 Ingeniería de Software, Guayaquil-Ecuador, Octubre-2003, pp. 1-15

[2] Jorge Mazón, José Alvear, Gipsy Bracco, Mónica Villavicencio. "Aspectos de la calidad y dificultades en la gestión de proyectos de software: "Estudio exploratorio"", ESPOL – VLIR, Componente 8 Ingeniería de Software, Guayaquil-Ecuador, Septiembre-2005, pp. 1 - 12.

[3] M. Villavicencio, J. Mazón, J. Alvear, "Elaboración y análisis de métricas para el proceso de desarrollo de software, "aplicado a pymes ecuatorianas desarrolladoras de software" Proceedings de las Terceras Jornadas de Ingeniería de Software ESPOL 2006, Guayaquil-Ecuador, Octubre, pp. 1-10.

[4] J. Mazón, J. Alvear, "Elaboración y análisis de métricas para el proceso de desarrollo de software para empresas desarrolladoras de software del ecuador" Plan piloto 2006, TESIS ESPOL, Guayaquil-Ecuador, 2006.

[5] Raúl González, Henry Hernández, "Desarrollo de un código de métricas para empresas desarrolladoras de software ecuatorianas pequeñas", TESIS ESPOL, Guayaquil-Ecuador, 2006.

[6] Martínez E. Medición. "Requisitos y procedimientos para construir un instrumento de medición.", Universidad Metropolitana, Venezuela, 2004, pp. 1 - 6

[7] Instituto Argentino de normalización y certificación " Orientación acerca del enfoque basado en procesos para los sistemas de gestión de calidad", Argentina, Mayo 2001 , pp. 1 – 12

[8] Joaquín García, "CMMI – CMMI nivel 2", España ,Noviembre 2005, pp. 1-5

[9] Horngren-Sundem-Stratton, "Contabilidad Administrativa", México, Prentice Hall, 2001.

[10] Microsoft, "¿Por qué su empresa no se puede permitir una mala documentación?", disponible en http://www.microsoft.com/spain/empresas/rrpp/mala_documentacion.mspx, última visita: Octubre 2006

[11] Corporación de Promoción de Exportaciones e Inversiones - CORPEI, "Condiciones del mercado laboral", disponible en, www.ecuadorinvest.org/ecuadorinvest/docs/10_8Condiciones_del_Mercado_Laboral.pdf, última visita: Octubre 2006.

A Organização de uma Máquina de Processo e a Melhoria do Processo de Produção de Software em um Ambiente de Fábrica

José A. Fabri[123], André L. P. Trindade[13], Alexandre L'Erário[12], Marcelo S. de P. Pessoa[1]
[1]*Departamento de Engenharia de Produção da Escola Politécnica – Universidade de São Paulo,*
[2]*Fundação Educacional do Município de Assis,* [3]*Faculdade de Tecnologia de Ourinhos*

Resumo

A organização de uma máquina de processo é um dos fatores relevantes para a concepção de uma fábrica de software. A máquina quando organizada deve dar subsídios para a administração de todo o ciclo de vida do componente. Ao organizar e institucionalizar uma máquina, a fábrica pode obter não só a melhoria do processo de produção de software, mas sim a institucionalização do mesmo. Este artigo apresenta a organização de uma máquina de processo e a valida-a por meio de experimento controlado.

1 Introdução

Atualmente, o mercado de desenvolvimento de software brasileiro trava uma batalha constante na busca pela qualidade e produtividade. Esta informação pode ser comprovada ao analisar-se os vários programas de incentivo promovido pelo Ministério de Ciência e Tecnologia (MCT), através da Secretaria de Política em Informática (SEPIN), onde o governo estabeleceu na Política Industrial Tecnológica que Software é um dos quatro temas prioritários (Software, Semicondutores, Indústria de Base e Fármacos). Entre tais programas, é possível destacar o SOFTEX (Sociedade para Promoção da Excelência do Software Brasileiro), cujos objetivos são: situar o Brasil entre os 5 (cinco) maiores produtores e exportadores de software do mundo e alcançar padrão internacional de qualidade e produtividade.

Além destes programas, o MCT e a SEPIN desenvolvem, periodicamente, uma pesquisa para verificar atributos de qualidade e produtividade do mercado brasileiro de desenvolvimento de software. A última delas foi publicada em 2006 e, ao efetuar uma análise nos dados, é possível chegar às seguintes conclusões: no Brasil existem cerca de 11.000 empresas com atividades relacionadas ao desenvolvimento e comercialização de software, estas empresas empregam 158.353 pessoas. Cerca de 25% das empresas possuem um programa de qualidade definido, outras 26% sentem a necessidade de estabelecer um programa de qualidade, isso comprova que o mercado brasileiro está tomando consciência da necessidade da qualidade em seus produtos (contexto software) MCT-SEPIN (2002) [10].

Paralelamente, a tais fatos, existe uma discussão, nos âmbitos empresarial e acadêmico, sobre o tema "fábrica de software": muitas empresas classificam o processo de desenvolvimento de software convencional como sendo fabril; a maioria destas empresas não possui um processo que prime por produtividade e qualidade e; um processo de desenvolvimento que não atenda a estas características não pode ser considerado fabril.

Costa (2003) [3], apresenta uma pesquisa envolvendo 31 empresas, as mais significativas, que atuam no mercado brasileiro utilizando o modelo de Fábrica de Software. Destas, apenas 41% aplicam um ciclo completo de desenvolvimento de software para seus produtos; 45% aplicam metodologia própria; 16% utilizam ferramentas de controle de projetos; 14% possuem certificação CMMI ou ISO; 13% utilizam ferramentas CASES e 10% aplicam métricas de qualidade.

Há, evidentemente, no mercado confusão com relação à compreensão do que seja Fábrica de Software. Pode ser feito um paralelo com a manufatura na qual Fábrica trata da produção repetitiva de um determinado produto. O conceito de componente é largamente utilizado nesta área. Uma montadora, como diz o nome, monta componentes fabricados por terceiros para produzir veículos: sistema de freios, sistema de injeção, suspensão, entre outros. Na indústria eletrônica o mesmo é aplicado em vários níveis de abstração: o componente, circuito integrado, é montado em uma placa de circuito impresso que forma um componente padronizado, por exemplo: uma placa de assinante. A placa é montada em um bastidor formando, assim, a central telefônica que possui configuração exclusiva para

cada fornecimento de cada cliente. Portanto, embora cada central telefônica seja única, suas partes são formadas por componentes padronizados obtidos no mercado ou construídos, internamente.

Este trabalho foi desenvolvido com base no contexto acima e com o objetivo de relacionar a organização de uma máquina de processo à melhoria do processo de produção de software, em um contexto de fábrica. Para atingir o objetivo proposto, o trabalho apresenta: na seção 2, a formalização do conceito de fábrica de software; na seção 3, são relatados os conceitos inerentes a componentes de software (teoria que dá subsídios para que a máquina possa ser implementada); a formalização da máquina é apresentada na seção 4; um experimento controlado da organização da máquina é relatado na seção 5 e; por fim, a seção 6 apresenta a relação entre a organização da máquina e a melhoria do processo de software.

2 Formalização Conceitual sobre Fábrica de Software

Na literatura, é possível encontrar vários autores que trabalham, diretamente, com o conceito de fábrica de software, dos quais destacamos as principais definições.

Segundo Cusumano (1989) [4], o termo fábrica de software foi utilizado, pela primeira vez, na década de 1960, no Japão. Várias empresas associam o termo ao mero desenvolvimento de software; entretanto, empresas que não atendam características como: produção em larga escala; padronização de tarefas; padronização de controle; divisão do trabalho; mecanização e automatização, não podem ser consideradas fábricas de software. Para o autor, o desenvolvimento de uma fábrica implica nas boas práticas da engenharia de software aplicadas, sistematicamente.

Já Bemer (1969) [2] define fábrica de software como um ambiente no qual se constrói programas e se efetuam testes. Neste ambiente devem existir ferramentas para realizar as ações de construir e testar. Uma fábrica deve possuir medidas de produtividade e qualidade, os registros financeiros devem ser mantidos por custo da programação e a forma de gerenciamento deve dar subsídios à previsão e à estimativa de dados de futuros projetos.

Para Basili *et. al.* (1992) [1], uma organização com características de fábrica de software deve possuir uma estrutura de construção de software baseada em componentes. Os componentes utilizados podem ser desenvolvidos pela fábrica de componentes (ou unidade de produção de componentes). Através desta definição pode-se concluir que uma fábrica de componentes é a base para a implementação de uma fábrica de software.

Segundo Li *et. al.* (2001) [9], uma fábrica de software deve possuir: um conjunto de ferramentas padronizadas para a construção do produto; bases históricas para o gerenciamento de projetos e; principalmente, um alto grau de reuso de código no processo de produção de um determinado software.

Fernstrom *et. al.* (1992) [7] definem que uma organização fabril para o desenvolvimento de software deve estar calcada na questão “única do software”, isto é, todo software é único, porém partes individuais são repetidas em vários projetos. O conceito fabril deve alicerçar o desenvolvimento, armazenamento e montagem das partes repetidas em um produto único.

Baseado nas definições apresentadas, este trabalho adota fábrica de software como uma organização estruturada, voltada para a produção do produto software, totalmente alicerçada na engenharia e com organização do trabalho, modularização de componentes e escalabilidade produtiva caracterizada. Deve possuir, ainda: um ambiente de gerenciamento de projetos; um processo padronizado, definido e institucionalizado; políticas que garantam a qualidade do produto; um conjunto de ferramentas para mecanizar gerenciamento de projeto, processo e construção; técnicas para medir e estimar custo, prazo e tamanho de uma equipe para um determinado projeto; ambiente de teste definido e padronizado; foco em um segmento de mercado e; política de desenvolvimento de recursos humanos (Fabri *et. al.* (2003) [6]).

3 A Teoria de Componentes de Software Subsidiando a Organização de uma Máquina de Processo.

Schaefer et. al. (1999) afirmam que a máquina de processo, pode ser classificada como um software com o objetivo de auxiliar na comunicação e coordenação das atividades realizadas pelos envolvidos no processo.

Tendo em vista que as atividades do processo de produção de software possuem em sua concepção a idéia de componentes ou artefatos de entrada e resultantes. É válido afirmar que tal máquina, ao coordenar as atividades realizadas, está provendo um mecanismo de organização do trabalho que provê informações sobre o que foi produzido e por quem.

De posse do fato expresso, anteriormente, os autores deste trabalho, encaram a máquina de processo como uma estrutura sistêmica de organização do trabalho, materializada em software, com o objetivo de armazenar e recuperar informações gerenciais relacionadas à produção de componentes ou artefatos dentro do contexto processual para software. Isto levou os autores a relacionar a formalização do objeto de pesquisa (a máquina) proposto com a teoria de componentes de software.

Segundo Greenfield (2003) [8], o desenvolvimento baseado em componentes é uma das técnicas que as fábricas de software empregam. O uso desta técnica de desenvolvimento associado à elaboração de linhas de produtos reduz, significantemente, o redesenvolvimento de linhas de código. Para que o reuso seja eficiente em um ambiente de fábrica, os arquitetos procuram expressar os relacionamentos entre grandes unidades de design, tais como *web services* e componentes de negócio. As técnicas de desenvolvimento em componentes envolvem composição e decomposição, além disso, o arquiteto tem o desafio de mapear as funcionalidades desejadas sobre os componentes.

Segundo Wang (2000) [12], os componentes são blocos de construção básicos, que alicerçam a engenharia de software baseada em componentes. A granularidade destes blocos implica na reusabilidade, produtividade e manutenção dos componentes. Um componente de software é um conjunto de código que encapsula uma ou mais funcionalidades.

Para Yacoub (2005) [13], os componentes de software têm três descrições básicas: informal, externa e interna. A descrição informal busca responder as questões relacionais entre o homem e o componente. A descrição externa define, formalmente, a comunicação entre o componente e a plataforma por este utilizado. Finalmente, as características internas refletem aspectos de implementação (quando se trata de código) e conceptivos (quando se trata de ativos de processo ou artefatos) do componente.

Já Doucet *et. al.* (2002) [5] salienta que um componente de software é uma unidade de composição que pode assumir a forma de uma função, de um objeto, de uma biblioteca ou de um programa completo. Os autores propõem a classificação dos componentes em níveis, segundo a sua abstração.

Este trabalho parte dos mesmos princípios propostos por Doucet et. al (2002) [5] e Wang (2000) [12] e, também, classifica os componentes em níveis, conforme mostra a Figura 1, na qual é possível verificar a distribuição dos níveis "específico", "comum" e "infra-estrutura" pelos eixos "dependência do domínio" e "granularidade" e constatar que quanto maior a granularidade, maior o grau de reutilização do componente. Por exemplo: Os componentes classificados como "infra-estrutura" fazem parte da composição da maioria dos softwares de uma determinada empresa. Por outro lado, os componentes "específicos" possuem um alto grau de "dependência do domínio", conseqüentemente, baixo grau de reutilização.

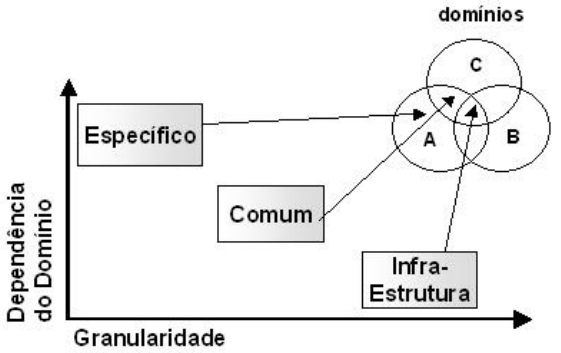


Figura 1: Níveis de Abstração de um Componente em Relação aos Domínios da Aplicação

Um fato de extrema importância apresentado por Doucet *et. al.* (2002) [5], é a necessidade de integração dos componentes, dada a sua granularidade. Os autores salientam que a integração de componentes pode ser feita de três formas:

- **Programada**: O "integrador de componentes" deve desenvolver um programa que compõe dois ou mais componentes;
- **Gráfica**: A composição dos componentes se dá por meio de um aplicativo ou ferramenta. Integração esta que provê maior grau de automação no processo de composição e, conseqüentemente, maior produtividade;
- "***Script***": Necessidade de desenvolvimento de um "*script*" para a composição de dois ou mais componentes.

Complementando a definição de componentes proposta por Doucet *et. al.* (2002) [5], Seaman (1992) [11] e Basili *et. al.* (1992) [1] relatam que um componente não pode ser classificado somente como código, para eles os componentes podem ser classificados como formulários, *templates*, documentos (artefatos) e ativos de processo.

Por fim, um fator que irá interferir, diretamente, na organização da máquina é a proposta de

representação de um componente, elaborada por Wang (2000) [12]. O autor salienta que um componente de software deve ser representado por uma "*sextupla*" interativa[1]:

(0) *C = {Q, I, f, O, q0, F}*

onde *C* representa o componente em questão, *Q* traduz o conjunto de estados de um componente, *I* representa o conjunto de entradas, *f* provê a função de transformação de um componente, *O* denota o conjunto de saída, *q0* apresenta o estado inicial e, por fim, *F* representa o estado final do componente.

4 Formalização da Máquina de Processo para Fábrica de Software

A principal função de uma máquina de processo é prover mecanismos de armazenamento e recuperação de informações a eles relacionados. Estas informações devem dar subsídios para que o "gerente de projetos" possua um controle rela de produtividade e qualidade.

Partindo da necessidade de armazenamento, recuperação e gerenciamento de informações, e que todo componente tem um ciclo de vida definido (Sommerville (2003)), foi inferido, neste trabalho, um conjunto básico de informações, para configuração da máquina de processo, relativas:

- à **gestação** do componente (o projeto de concepção do componente);
- ao **nascimento** do componente (a implementação ou o instanciamento da versão 1);
- à **evolução** do componente (o crescimento), a qual também se dá pelo desenvolvimento das atividades de implementação e testes. Novas versões também configuram o crescimento do componente;
- à **reprodução** do componente (um componente *a* se unido a um componente *b* resulta em um componente *c*);
- à **morte** do componente.

Com base nas informações acima é possível verificar que a máquina de processo pode ser representada pela seguinte "*quíntupla*" (alusão a Wang (2000) [12]):

(1) *B = {G, N, E, R, M}*

onde *B* armazena o nome da máquina representada, *G* armazena informações sobre a gestação de um componente, *N* armazena informações relacionadas ao nascimento, *E* trata informações sobre a evolução, *R* está relacionada com a reprodução e, por fim, *M* relata se um componente está em desuso ou morto.

Todas as variáveis, com exceção da variável *M*, da "*quintupla*" *B* derivam novas tuplas (derivação esta definida segundo as necessidades de customização da máquina), representadas a seguir:

(2) *G = {i, p, C}*

onde *i* armazena as informações da instituição que direcionou a concepção de um determinado componente, *p* retrata as informações relacionados ao projeto para o qual o componente foi concebido e *C* apresenta as informações de um determinado componente (informações estas apresentadas por Wang (2000) [12], na seção 3).

(3) *N = {C, v,l}*

onde *v* representa a versão de um determinado componente, o nascimento se dá após o estabelecimento da versão 1 e *l* armazena o local de armazenamento do componente. É importante salientar que as informações dos componentes são configuradas na gestação e no nascimento, fato que levou a inserção de *C* na tupla *N*.

(4) *E = {a,t}*

onde *a* armazena as informações relacionadas a atividade versus tempo, por exemplo: um componente levou 15 horas (tempo) na atividade de produção. Já *t* armazena informações relativas o teste.

(5) *R = {c,h}*

onde *c* armazena as informações de quais componentes foram compostos para a geração de um novo componente e *h* retrata a forma de composição de um componente que, segundo Doucet *et. al.* (2002) [5], pode ser classificada como programada, gráfica ou *script*.

Por fim, na seção 5, será apresentado um experimento controlado da organização de uma máquina de processo, baseado na formalização apresentada nesta seção.

5 Organização de uma Máquina de Processo para Fábrica de Software: Um Experimento Controlado

Antes de apresentar a organização da máquina de

[1] Ressalta-se que a "*sextupla*" não possui relação com o conceito de máquina de estados, ela é, simplesmente, uma notação formal utilizada na representação interativa de um componente.

processo, faz-se necessário apresentar o ambiente fabril na qual a mesma foi desenvolvida.

O ambiente utilizado para a organização da máquina, pertence à fábrica de software do Departamento de Engenharia de Produção, da Escola Politécnica da Universidade de São Paulo, ao Centro de Pesquisas de Informática (CEPEIN) da Fundação Educacional de Município de Assis e a Fábrica de Software de Faculdade de Tecnologia de Ourinhos. No ambiente em questão um processo com características fabris está sendo implementado. A Figura 2 apresenta uma visão geral do processo de produção, no qual é possível verificar a presença de 5 atividades: Negociação, Gerenciamento de Projeto, Construção, Teste e Entrega.

Na atividade de negociação o diretor da fábrica de software fecha os contratos para o desenvolvimento dos produtos. Esta atividade é extremamente importante, pois ela proporciona a relação direta entre fábrica e clientes.

A atividade de gerenciamento de projeto tem, como objetivos, planejar e controlar a execução do projeto, dentro daquilo que foi planejado. Esta atividade possui as seguintes tarefas: definir o escopo do produto a ser gerado; definir e seqüenciar atividades do projeto, inserindo-as na máquina (esta definição deve respeitar as atividades do processo); definir cronograma, segundo as atividades, previamente, seqüenciadas na tarefa anterior e; estabelecer os recursos necessários para a execução do projeto (recursos humanos, por exemplo). É importante salientar que a atividade de gerenciamento de projeto ocorre em paralelo às demais atividades do processo.

A atividade de construção possui as tarefas: 1 - Verificar se o documento de arquitetura, advindo de algum dos possíveis clientes da fábrica de software, segue o padrão pré-estabelecido em contrato. 2 - Verificar se os componentes existentes na base atendem os requisitos materializados no documento de arquitetura. Se atenderem, é necessário configurar a ordem de montagem (esta ordem expressa quais são os componentes necessários para que uma funcionalidade do software possa ser implementada). Caso contrário, o componente deve ser desenvolvido. 3 – Realizadas estas tarefas, o software entra em produção (tarefa montar software). Ressalta-se que todas informações geradas nesta atividade são inseridas na máquina de processo.

Na Figura 2 também é possível verificar que as tarefas “desenvolver componentes” e “montar software” disparam a atividade de teste. O teste, na fábrica de software, pode ser realizado de várias formas, entre elas destaca-se o teste unitário (teste de um único componentes), o teste de integração (vários componentes interligados para prover uma funcionalidade do software) e o teste de aceitação do produto (software ou componente), teste este realizado na atividade de entrega.

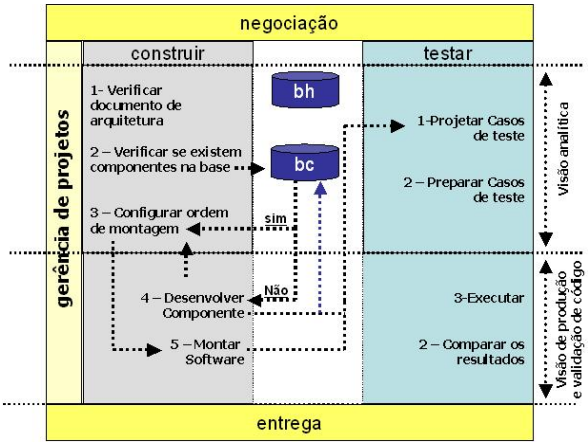


Figura 2: Processo Fabril de Desenvolvimento de Software (implementado junto ao CEPEIN). Legenda de Figura (bc: base de componentes; bh base histórica de projetos)

É possível perceber, na Figura 2, que a fábrica de software desenvolvida possui duas visões, *analítica* e *produção e validação de código*.

Na visão analítica, os envolvidos com o processo de produção analisam documentos, consultam a base de componentes de código, projetam casos de testes e preparam os dados para o mesmo. Nesta visão não é feito qualquer tipo de codificação.

Na visão de produção e validação de código, os programadores e testadores desenvolvem componentes, montam softwares, executam os testes e comparam os resultados (validação de produto, segundo as especificações).

Além do processo, o ambiente possui uma série de ferramentas (geradores de código, automação dos calculo de métricas), que auxiliam os envolvidos com o processo de produção de software a desenvolver suas tarefas.

Após esta breve descrição sobre o ambiente (processo e ferramentas), os próximos parágrafos apresentarão a organização da máquina de processo relacionando-a aos conceitos apresentados nas seções 3 e 4.

A organização da máquina de processo segue todo o modelo de tuplas apresentado na seção 4, esta informação pode ser constatada na Figura 3. Ao

analisar esta figura é possível constatar a correlação entre o modelo de dados da máquina e as tuplas *G*, *N*, *E*, *R* e a variável *M*.

Na tupla *G* é possível verificar a presenças das informações dos projetos (*p*), das instituições (*i*) e parte das informações dos componentes (*C*).

É possível verificar, na tupla *N*, todas as variáveis para representação de um componente propostas por

Wang (2000) [12]. Além desta representação, dados como: código da versão, código do componente, data do início e término do desenvolvimento e local de armazenamento da versão, foram inseridos nesta tupla.

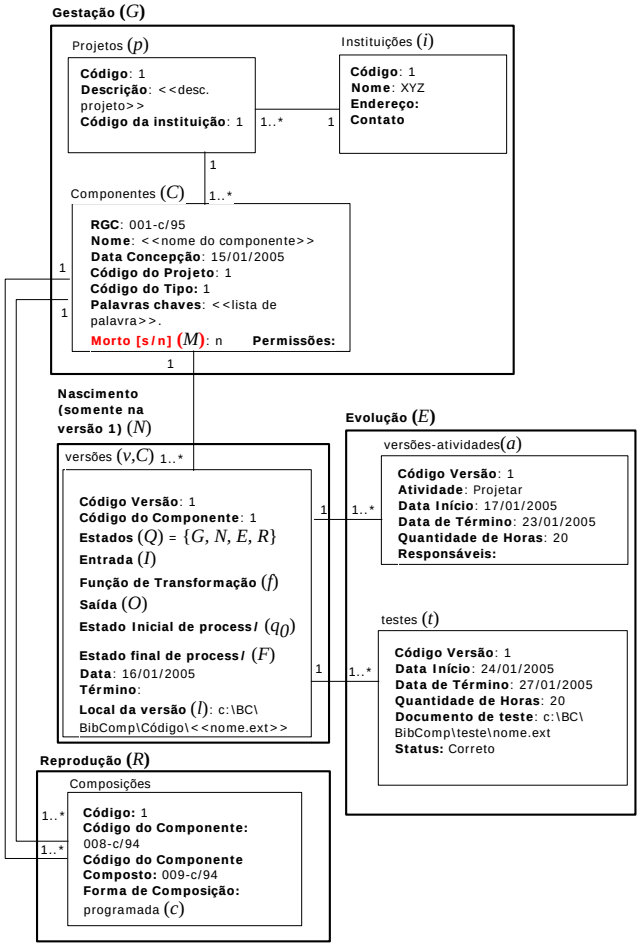


Figura 3: Organização da máquina de processo

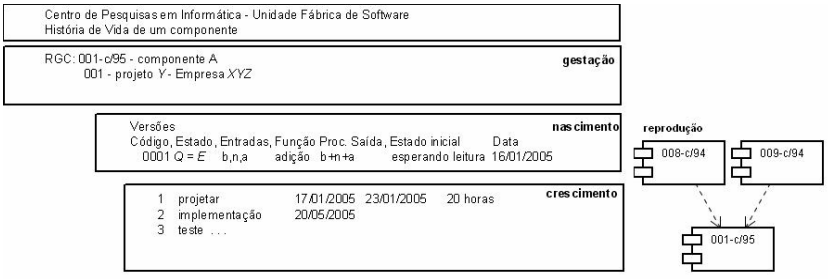


Figura 4: História de Vida do Componente.

As informações relacionadas às atividades versus tempo podem ser verificadas na tupla *E*, por exemplo: o componente 001-c/95 na versão 1 permaneceu 20 horas na atividade de projeto. As informações sobre a atividade de teste também estão relacionadas à tupla *E*, fato este que pode ser comprovado na Figura 3.

A tupla *R* retrata que o componente 001-c/95 é resultado da união dos componentes 008-c/94 e 009-c/94.

Por fim a variável *M* (que representa a morte do componente) está representada em vermelho na Figura 4, indicando que o componente está "vivo".

Analisando ainda a Figura 3 é possível verificar que o "gerente de projeto" tem acesso a toda "história de vida" de qualquer componente. Uma das possíveis informações que a máquina fornece pode ser verificada por meio da Figura 4.

Na Figura 4 é perceptível que: o "*componente A*" que possuir "Registro Geral (*RGC*) = *001-c/95*"; foi "concebido" pelo "projeto *Y*" da "empresa *XYZ*"; o componente "nasceu" em *16/01/2005* e iniciou seu "crescimento" no dia *17*, e o mesmo continua em "evolução" (*Q = E*). O componente *001-c/95* é "filho" dos componentes *008-c/94* e *009-c/94*.

Esta seção apresentou a máquina de processo de forma organizada. A seção 6 proverá a relação entre tal organização e a melhoria do processo de software.

6 A Organização de uma Máquina de Processo como Ingrediente na Melhoria de um Processo de Software[2]

Um dos primeiros sinais de melhoria do processo de software, relacionado à customização e à utilização de máquina de processo, está intimamente ligado à definição das atividades do processo para a produção de um componente, fato este que pode ser verificado no quadro versões-atividades da Figura 3.

Definidas as atividades do processo, um determinado componente irá percorrê-las durante sua evolução (*E*). Ao percorrer tais atividades, os envolvidos com o processo de software devem informar as datas de início e término e a quantidade de horas que um determinado componente permaneceu em uma determinada atividade. De posse destas informações, o gerente de projeto pode verificar se o desenvolvimento do produto está respeitando questões relativas a cronograma e custo. Fato este que também pode melhorar o processo de software de uma determinada empresa ou fábrica.

Um outro aspecto interessante, em relação à melhoria do processo de software, quando a máquina de processo é customizada e utilizada, é o das métricas de software: As informações relativas à quantidade de horas que um componente levou para ser desenvolvimento podem ser cruzadas com a quantidade de pontos por função (ou outra unidade de medida qualquer) resultando, assim, em uma média de produtividade (vide quadro evolução da Figura 3)

Ao utilizar a máquina, um documento de teste deve ser configurado. O mesmo deve ser formalizado e customizado segundo as necessidades da empresa ou fábrica de software, ação esta traduzida em uma melhoria do processo de teste.

O emprego de uma máquina sistemática auxilia o gerenciamento de configuração, mantendo a integridade entre os produtos de trabalho. O gerenciamento de configuração implica em determinar e manter uma *baseline*; estabelecer políticas de controle de acesso e armazenar solicitação de modificações.

Com o passar do tempo e com as informações inseridas na máquina de processo, uma base histórica de projetos é formada, fato este comprovado ao analisar a arquitetura organizacional da máquina. Esta arquitetura pode prover informações do tipo: O projeto *X* com características *Y* possui *Z* componentes de tipo *K* (caso as informações de tipos de componentes sejam inseridas na máquina). O tempo de desenvolvimento do projeto *X* também é uma informação que pode ser inferida na máquina. Quando um novo projeto (denominado *J*), com características semelhantes a *X*, for desenvolvido, o gerente de projeto terá subsídios mais consistentes para prever custo e prazo para *J*.

Enfim, algumas boas práticas dentro do contexto de produção de software são iniciadas se a máquina para armazenamento de componentes for customizada e, principalmente, utilizada de forma correta.

7 Conclusões

Este trabalho apresentou a relação entre a organização de uma máquina de processo e a

[2] As informações apresentadas nesta seção foram inferidas a partir de estudos de casos realizadas a 6 fábricas de softwares brasileiras (todas com cerificação de qualidade CMMI nível 2, no mínimo). Ressalta-se que todas elas possuem em sua estrutura organizacional uma máquina de processo implementada.

melhoria do processo de produção de software em um ambiente fabril.

No trabalho, foi possível verificar que a organização de uma máquina de processo não demanda um grande esforço. Toda empresa que trabalha com a atividade de desenvolvimento de software pode organizá-la, customizá-la, implementá-la (basta possuir uma ferramenta para gerenciamento de banco de dados) e, principalmente, institucionalizá-la.

A visão da máquina apresentada está customizada para uma organização com características de micro empresa, porém, o mesmo pode ser feito para grandes empresas.

Enfim, a organização, a customização e, principalmente, a utilização coerente da máquina podem trazer a uma empresa de software a melhoria de seu processo de produção. A máquina quando utilizada provê condições para: Estabelecimento das atividades do processo (possibilidade de institucionalização do processo); Gerenciamento de projeto; Delimitação de métricas de produtividade; Gerenciamento de configurações; Desenvolvimento da uma base histórica de projetos.

Referências Bibliográficas

[1] Basili, V. R.; Caldiera, G.; Cantone, G.; A Reference Archiecture for the Component Factory; in: *ACM Transaction on Software Engineering and Methodology*. vol 1. nº 1. pp 53-80. January 1992.

[2] Bemer, R. W.; The Economics of Program Production**;** In: *Information Processing* - vol.II, nº 68; Amsterdan: North-Holland Publ.Co, 1969.

[3] Costa, Ivanir; *Contribuição para o aumento da qualidade e produtividade de uma fábrica de software através da padronização do processo de recebimento de serviços de construção de softwares* - 174 pag.; Tese (Doutorado) Apresentada ao Departamento de Engenharia de Produção da Universidade de São Paulo; São Paulo: PoliUSP, 2003.

[4] Cusumano, Michael A.; Software Factory: A Historical Interpretation; in: *IEEE Software* - vol. 6, Nº 2, pp. 23-30. March/April 1989.

[5] Doucet, F.; Shukla, S.; Gupta, R.; Otsuka, M.; An Environment for Dynamic Component Composition for Efficient Co-Design; In: *Proceedings of the Conference on Design, Automation and Test in Europe* ; Washington, DC: IEEE Computer Society , March, 04 - 08, 2002.

[6] Fabri, J. A.; Trindade, A. L. P.; Begosso, L. R.; L'Erário, A.; Silveira, F. L. F.; Pessôa, M.o S. de P.; Techniques for the Development of a Software Factory: Case CEPEIN-FEMA; in: *Annals of 17th International Conference Software & Systems Engineering and their Applications*; Paris: CNAM, December 3, 2004.

[7] Fernstrom, C. Narfelt K. H. Ohlsson L. Software Factory Principles, Architecture and Experiments. *IEEE Software*. (Vol. 9, No. 2) pp. 36-44. March/April 1992.

[8] Greenfield, J. and Short, K.. Software factories: assembling applications with patterns, models, frameworks and tools. In *Companion of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*. Anaheim, CA, USA, October 26 - 30, 2003. OOPSLA '03. ACM Press, New York, NY. 2003.

[9] Li, C.; Li, H.; Li, M.; A Software Factory Model Based on ISO 9000 e CMM for Chinese Small Organization; in: *Second Asia-Pacific Conference on Quality Software (APAQS'01)*; Hong Kong: IEEE, December, 2001.

[10] MCT-SEPIN- Secretária de Política de Informática do Ministério da Ciência e Tecnologia; *Relatório de Qualidade e Produtividade no Setor de Software Brasileiro N4* - ISSN 1518-112X; Brasília: MCT, 2002.

[11] Seaman, C. B.; AAA: a modeling language for software production environments; in: *Proceedings of the 1992 Conference of the Centre For Advanced Studies on Collaborative Research* – Vol. 1. J. Botsford, A. Ryman, J. Slonim, and D. Taylor, Eds. IBM Centre for Advanced Studies Conference. IBM Press, 513-530.

[12] Wang, J. A.; Towards component-based software engineering; in: *Proceedings of the Eighth Annual Consortium on Computing in Small Colleges Rocky Mountain Conference*; Orem, United States: Consortium for Computing Sciences in Colleges, 2000.

[13] Yacoub, Sherif. Ammar, Hany e Mili, Ali. *Characterizing a Software Component*. http://www.sei.cmu.edu/cbs/icse99/papers/34/34.htm, acessado em julho de 2005.

A minimal OCL-based Profile for Model Transformation

Roxana Giandini (1) (2) Gabriela Pérez (1) Claudia Pons (1) (3)
(1) LIFIA, Facultad de Informática,Universidad Nacional de La Plata, Buenos Aires, Argentina
(2) Dpto. Sistemas de Información, Facultad Regional La Plata,
Universidad Tecnológica Nacional.
(3) CONICET (Consejo Nacional de Investigaciones Científicas y Técnicas)
[giandini, gperez, cpons]@lifia.info.unlp.edu.ar

Abstract

The MDD (Model Driven Development) initiative covers a broad spectrum of research areas such as modeling languages, definition of transformation languages among models, and construction of support tools. Currently, some of these aspects are being established and applied, while others are still in the process of definition. Consequently, it is necessary to make every effort to convert MDD and its concepts and related techniques into a coherent proposal, based on open standards, and supported by mature tools and techniques.

Transformations among models require specific languages for their definition. These languages must have a formal base, for instance, a metamodel that supports them and allows for an automated treatment. This paper presents a declarative language for model transformations inspired on OMG (Object Management Group) standards. Our proposal is expected to be a minimal extension of the already existing OMG specifications, and it basically uses OCL (Object Constraint Language) language to specify transformation relations.

1. Introduction

In the MDD paradigm [1], models are thought to be the primary conductors in all software development aspects. A PIM (Platform Independent Model) is transformed into one or more PSMs (Platform Specific Model); hence, a specific PSM is generated for each specific technological platform. Model transformation is the MDD engine; models are no longer mere contemplative entities and become productive entities.

The MDD initiative covers a broad spectrum of research areas: modeling languages, definition of languages for model transformation, construction of support tools for the different tasks involved, application of concepts to development methods and specific domains, etc. Currently, some of these aspects are well-based, and are being applied with some success; however, other aspects are still undergoing their definition process. In this context, it is necessary to make every effort to convert MDD and its concepts and related techniques into a coherent proposal, based on open standards, and supported by mature tools and techniques. Model transformations require specific languages for their definition; these languages should have a formal base, for example, a metamodel that supports them and allows for an automated treatment.

In this paper, we present a pure declarative language to express transformations among models whose initial metamodel is inspired in the QVT (Query\View\Transformation) language [4], which is the OMG [2] specification for transformations, still undergoing a definition process. The language we propose aims at being minimal for expressing queries and transformations among models.

The organization of this paper is as follows: in Section 2 the concept of model transformation is introduced by an example specified in QVT. Section 3 presents and discusses some disadvantages about QVT notation, thus informally realizing our proposed statement redefining the example. Section 4 presents a step-by-step construction for a transformation profile, based on the definition of stereotypes and the use of OCL language giving thus a formal base to our proposal. In Section 5, this profile is applied to the example. Finally, conclusions and future working lines are presented.

2. A Model Transformation example

For a better understanding of the model transformation notion, we present a simple example extracted from QVT manual. This example shows the textual specification of a transformation called UML2Rdbms that defines a relation Class2Table that in turn transforms UML classes (which fulfill the

constraint of being persistent, as indicated in the when clause of this relation) into Relational Model tables. For each class a corresponding table under the same name must exist. Also the transformation presents another relation: the Attr2Col, which specifies that the class attributes are in agreements with the columns under the same name in the corresponding table (the relation only transforms not multivalued attributes whose type is a basic data type). This relation is invoked in the where clause of Class2Table and it means that each time Class2Table is fulfilled for a class and a table, Attr2Col for its attributes and columns, respectively, are also fulfilled:

Transformation UML2Rdbms (Uml: UML2.0, Rel: RDBMS) {

TopLevel **Relation** Class2Table {
checkonly domain Uml c: Class {name = n }
checkonly domain Rel t: Table {name = n }
when { isPersistent = true }
where { Attr2Col (c, t) } }
Relation Attr2Col {
checkonly domain Uml c: Class { attribute = a: Attribute {name = an, type = p: DataType {name = dt}} }
checkonly domain Rel t: Table {column: col:Column {name = an, type.name = dt } }
when { not a.isMultivalued() }
where { col.type = a.type } }
}

A graphic instantiation of this transformation could be, for instance, the one shown in Figure 1 between two concrete models: Uml (from UML) and Rel (from Rdbms). In this case, for the class Person of the Uml model, there is a Table in the Rel relational model under the same name. Attributes undergo the same process: for each attribute of the class Person there is a column in the table Person under the same name and with the same type. That is to say, for the attribute `name´, there is a column `name´, and both are strings.

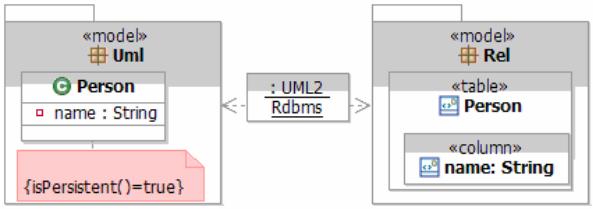


Figure 1. Transformation UML2Rdbms graphic instantiation.

3. Our proposal – Informal Introduction

In the previous example, certain issues that can be generally found in QVT examples could be observed. We will give some details of these issues over this particular example; they could be considered general disadvantages that hinder the QVT use and understanding.

Some of these issues are:

- Transformation relations are expressed through template expressions in each domain. Then, a pattern matching among these expressions will be performed to concrete the transformation. Due to the fact that in these expressions, variables can be recursively defined, this nesting and the consequent pattern matching could be broadcast producing complicated specifications.
- The relation Attr2Col suggests its application over all attributes of the Class parameter, but this is not explicitly indicated. Similarly, the same situation happens for the parameter Table. To denote that without ambiguity, we could use OCL language.
- Also, the same relation transforms specifically attribute to column, but its parameters are Class and Table. It would be clearer if each relation has as parameters the elements that actually transforms.

Consequently, we propose a new way to specify the same transformation considering the mentioned disadvantages, making more readable the notation, following a more friendly syntax and maximizing the OCL language use.

3.1 Redefining the example

In our proposal, each domain is specified with just the participating variables, while the conditions, that must be fulfilled among them, are specified in the where clause of the relation. It means, we are using neither template expressions nor pattern matching. Furthermore, in the relation Class2Table we explicitly specify in OCL the iteration over class attributes and we call the relation with the adequate parameters: Attribute and Column. Then, variables used in domain and codomain have respectively the proper type: Attribute and Column. Preconditions that attributes must fulfill, appear in the when clause, while post conditions are included in the where clause.

The example redefined in our notation is as follows:

Transformation UML2Rdbms (Uml: UML2.0, Rel: RDBMS) {

TopLevel **Relation** Class2Table {
checkonly domain Uml c: Class
checkonly domain Rel t: Table
when { c.isPersistent = true }

where { c.name = t.name and c.allAttribute-> forAll (a | (t.column -> exists (co | Attr2Col (a, co)))) } }

Relation Attr2Col {
checkonly domain Uml a: Attribute
checkonly domain Rel co: Column
when {not a.isMultivalued() and a.type.oclIsKindOf (DataType)}
where { co.type = a.type and co.name = a.name } }
}

In the next section we discuss an approach to define a formal base to the proposed notation. In order to realize this, a UML Infrastructure extension is constructed. This profile is based in stereotypes definition and maximizes the use of the OCL to avoid the creation of unneeded new elements. Stereotypes and constraints definition is the standard extension mechanism of UML. This mechanism promotes to extend existing metaclasses instead of defining new ones. In turn, it is commendable to minimize the amount of new stereotypes maintaining the simplicity and reducing the user's training time, encouraging the use of this profile.

4. A minimal OCL-based Infrastructure Extension for Model Transformation

Previously to present the analysis of how and where the transformation language is defined, let us review some concepts on techniques for metamodeling and definition languages.

4.1 Model Transformation in the 4-layer Metamodeling Architecture

Metamodeling is a mechanism that allows for the formal construction of modeling languages such as UML and RDBMS. The 4-layer Metamodeling Architecture is the OMG proposal aiming at standardizing modeling-related concepts, from the most abstract to the most concrete ones.

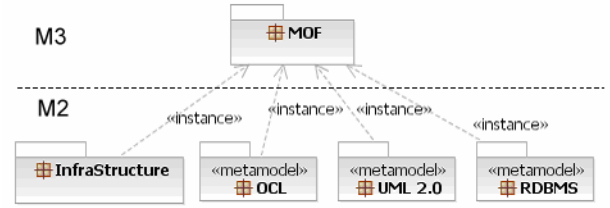


Figure 2. Layers M3 and M2 of the 4-layer Architecture

The levels defined in this architecture are commonly called: M3, M2, M1, and M0. Level M3 is the most abstract one, where MOF [3] is located; level M2 defines languages for specifying models, for instance UML; level M1 defines instances of a metamodel; Level M0 models the real system.

In this context, a specific transformation is located in layer M2 so as to be able to relate generic instances of concrete metamodels (located in M2), such as those of UML and RDBMS, while instances of these metamodels (for example a UML Class and an RDBMS Table) are actually transformed. That is to say, models that are concretely involved in the transformation (layer M1) are parameters for the transformation language.

It is reasonable to think that the metamodel for transformations and its instantiation cannot coexist in the same layer since they represent different levels of abstraction. Then definition of languages for model transformation may be realized in the layer M3 of the 4-layer Modeling Architecture.

However, MOF represents a close Meta-metamodel upon which metamodels (MOF instances) are instantiated and it is located in layer M3. Therefore, the transformation metamodel should be located in layer M2, together with the remaining metamodels (UML, OCL, etc.) as shown in Figure 2. Let us analyze other OMG specifications, closely related to MOF: one of these specifications is the Infrastructure [7] for modeling languages. The Infrastructure is the most simplified specification defining the basic constructors and the common concepts for modeling language. It could be said that Infrastructure is independent from the UML itself. The UML metamodel is complemented by the Superstructure [6] specification, that defines the constructors at UML 2.0 user's level.

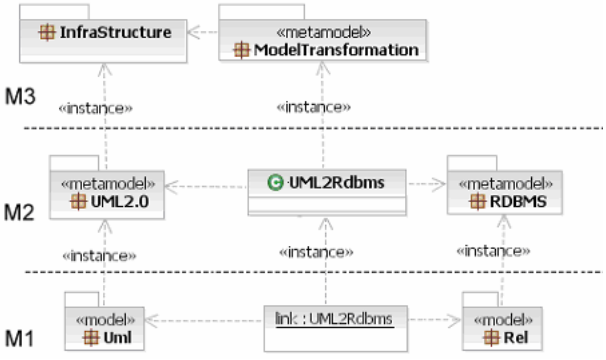


Figure 3. Model Transformation in 4-layer Architecture

The Infrastructure case is interesting for its recursive definition regarding MOF: on the one hand, it can be defined as a MOF instance (in layer M2 as shown in Figure 2); on the other hand, MOF itself is based on, or uses elements of the Infrastructure Core package for its definition, thus allowing for the identification of the Infrastructure as a meta-metamodel. If the new transformation metamodel is defined as an extension of

elements of the Core package, it is questionable to think that it lies in level M3 together with the Infrastructure (see Figure 3). Then, a transformation metamodel instance in layer M2 relates metamodel elements, while in layer M1 will be a correspondence link among models - instances of such metamodels, which are the ones that are concretely transformed.

In the QVT specification document, the metamodel is defined as a "MOF and OCL extension" [5]. Our point of view, following what has been earlier analyzed and the OMG definitions, is that it is not technically correct to extend MOF since it represents a close Meta-metamodel upon which metamodels are instantiated. Hence, we propose to minimally extend Infrastructure 2.0 and use OCL 2.0 to implement the transformation metamodel.

The QVT definition consists of three packages: two declarative packages (Relations and Core) and one imperative package. A large number of the languages proposed for model transformation [8, 9, 10, 11 and 12] are based on the QVT language. Most of these language proposals are hybrid, being both declarative and imperative. To define a transformation relation among models, it seems to be enough to specify it declaratively and then choose in which imperative language it could be executed. Just then, executable statements in agreement with the declarations should be written. Thus, we chose to propose a pure declarative language for transformations, whose initial metamodel is inspired in the QVT Relations package.

permitting us to express relations and queries of model transformation.

4.2 Construction of the minimal profile

Following what has been stated in the above section, our proposal is to extend Infrastructure, by creating a specific profile (instance of the Profile metaclass from the Infrastructure) capable of expressing particular features of model transformation.

The extension offers a formal base that reflects our textual notation proposed in section 3. It is defined in five stages, as suggested in other works defining profiles:

First stage: definition of the proposed metamodel

Based on the QVT Relations package metamodel, the metamodel proposed (see Figure 4) uses the OCL existing metaclasses and minimizes the metamodel proposed in our previous work [18]. Specifically speaking, the concrete syntax (BNF grammar) of that package was adapted with the purpose of simplify its use and understanding, while keeping the necessary basic concepts; eliminating several abstract classes (such as Rule, RelationDomain, Pattern, PatternDomain, etc.) and other concepts that appear in the transformation domain (such us Query and Helper) were taken into account.

In our metamodel, a transformation consists of typed

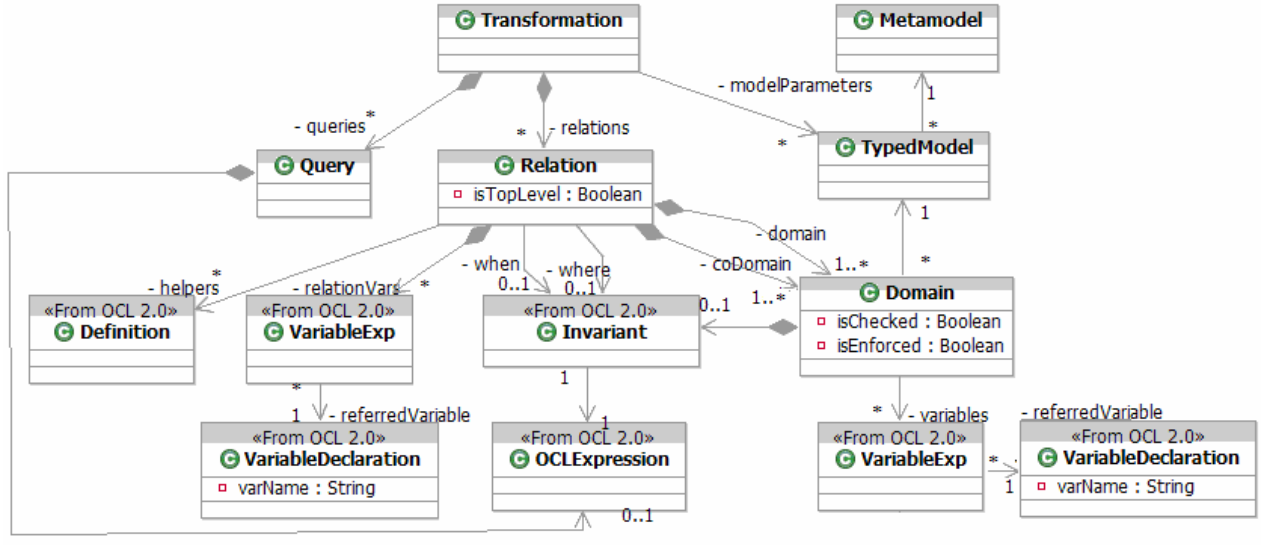


Figure 4. Metamodel proposed for Model Transformation

Then, we implemented it as an Infrastructure extension using the existing OCL metaclasses. This metamodel is expected to be minimal in the set of metamodels

models, relations and queries. Each typed model has a metamodel as type - any MOF instance – and it participates in the transformation as source and/or

target. A relation has a Boolean attribute isTopLevel, representing its level. A top level relation is executed automatically; otherwise the relation must be explicitly called by another one to be executed. Also relations may include variable declarations, when and where clauses - preconditions that should be fulfilled so as to achieve the transformation and postconditions that have to be fulfilled when finishing the transformation execution, respectively - and helpers. A helper, an element not taken into account in the QVT metamodel, defines additional operations in order to perform navigation upon the models participating in the transformation. Helpers could optionally have input parameters and use recursion. They consist of variable declarations and expressions that specify the additional operations definition. The application of a helper does not produce side effects.

Moreover, relations are defined from domains to codomains; each domain corresponds to a model and can only be checked or else, forced. When a domain is checked, its constraints will be evaluated regarding the current model-associated values. When a domain is forced, it is also checked, but in this case, the associated model can be updated (that is to say, object creation and/or erasure may occur) to realize the transformation.

In the QVT metamodel, TemplateExp metaclass represents complex expressions that consist in variable declarations and optionally other constraints. They are specified into the relation domains. The purpose of a TemplateExp is to arbitrarily define nested objects and/or collections templates for pattern matching and instantiation. In the QVT document the TemplateExp hierarchy is specify by new metaclasses added to the OCL metamodel.

To be able to understand the Template Expression issue, let us use the example in section 2. In the relation Attr2Col, the TemplateExp of the UML domain, is c : Class { attribute = a: Attribute, { name = an, type = p: DataType {name = dt}} } and the TemplateExp of the Rel codomain is t: Table { column: col:Column {name = an, type.name = dt }.

We propose to separate the variable definitions that could be nested, from the expressions which are used to match patterns in candidate models. For this reason, we define variables through the OCL Variable-Declaration metaclass, and the pattern matching expressions are added in the where clause. Specifically, to represent the variable definitions in domain and codomain, we propose to use the OCL VariableExp metaclass, an OCLExpression which consists of a reference to a typed variable (OCL VariableDeclaration metaclass) through the referredVariable association (see Fig. 4). Then, after analyzing the OCL 2.0, we conclude that maximizing the use of this language it is necessary to define neither new stereotypes nor new metaclasses to the proposed profile.

As far as queries are concerned, they specify instance transformations, in the same way as relations, but with a simpler format. The idea is that they are side-effect-free functions, with formal parameters and an OCLExpression as body.

Second stage: Metamodel stereotypes definition

For each metaclass of the step above, a new stereotype has been defined. Unlike in UML, in Infrastructure the stereotype's taggedValues are properties called ownedAttribute. The following subsection shows the new stereotypes with its extended metaclasses.

Regarding the relationships among the proposed metaclasses, we have decided not to stereotype them to avoid using an excessive notation when instancing the profile. We have chosen to add constraints to the connecting concepts. Association is used to relate

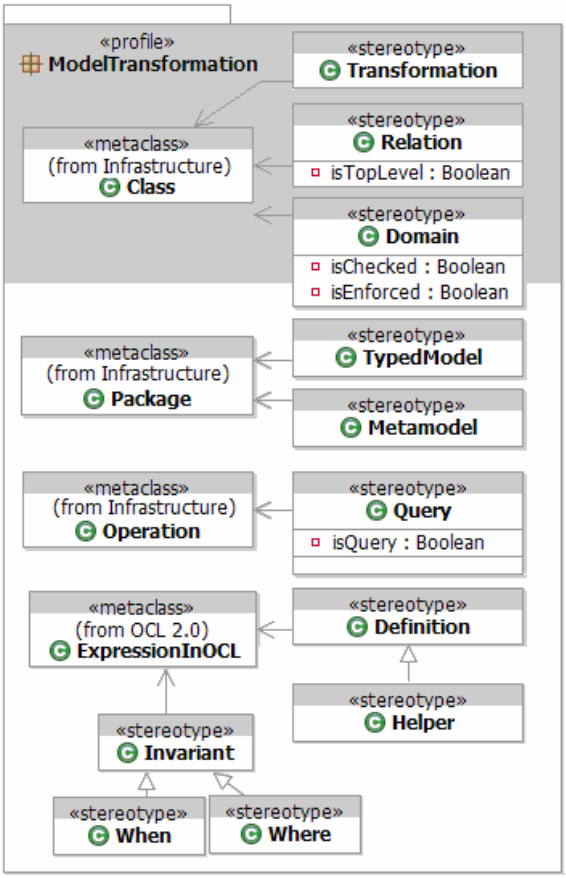


Figure 5. ModelTransformation profile

Classes, and DirectedRelationship[1] when relating Packages (as packages are not Classifiers, they cannot relate via Association).

Third stage: Identification of the proposed stereotypes's extende d metaclasses.

In this subsection, the Infrastructure metaclasses that could be extended to represent the proposed metamodel are identified.

According to this metamodel, a transformation could be defined as a Class formed by the models it transforms, and by the relations and queries defined among them. Hence, all models are referred to without indicating to which one transformation is not applied, nor in which direction it is performed. This declarative style best fits for our proposal. In this sense, a relation can also be represented extending the metaclass Class, made up of the intervening domains, and the conditions and helpers constraining it. Domain adds features to models and also has Class as extended metaclass.

The query stereotype extends the metaclass Operation. Its Boolean attribute isQuery already exists in Operation.

In general, models can be represented in packages. Therefore, TypedModel and Metamodel are defined extending the metaclass Package.

The relationship between transformation and query concepts is represented by defining the owner of the <<query>> operation as a <<transformation>> class.

Finally, let us analyze the expressions (when, where and helpers) which constrain a relation of our metamodel. In MOF-based modeling languages, in many of the cases where the term expression is applied, an expression written in OCL can be used. In the OCL abstract syntax, an OCLExpression is defined recursively. For this reason the metaclass ExpressionInOCL is needed to represent the root of this recursive notation. It has a body attribute (which would be the OCLExpression) and a language attribute which should have the "OCL" value for this case. An existing stereotype for an ExpressionInOCL is <<definition>>, which constrains the ExpressionInOCL indicating that it should be defined in the context of a Classifier; it could also define additional operations and have Let expressions. This makes an OCL <<definition>> expression appropriate for modeling helpers.

As regards the when and where clauses, they are Boolean expressions, therefore we specialized the existing <<invariant>> stereotype (which metaclass extended is ExpressionInOCL), for modeling them. An <<invariant>> ExpressionInOCL is an expression whose body is a Boolean OCLExpression constraining a Classifier. In our metamodel, these clauses constrain a <<relation>> class and can refer to other relations and helpers. Figure 5 shows the stereotypes defined with their extended metaclasses, making up the ModelTransformation profile.

Fourth stage: Stereotype attributes definition.

For each attribute in the metamodel proposed, an attribute is defined in the corresponding stereotype. Figure 5 shows the stereotypes with their attributes: IsTopLevel of Boolean type in Relation stereotype; isChecked, isEnforced, both of Boolean type in Domain stereotype. In Section 5, we present a transformation instantiation using the defined stereotypes and OCL.

Fifth stage: Complete definition of the proposed extension

The proposed extension is a Profile class instantiation within the Infrastructure 2.0 Profile package.

For those new stereotypes which add restrictions to its base metaclass, we present below a formal definition indicating which metaclasses they stereotype, their attributes and the constraints expressed in OCL that the extended metaclasses should be fulfilled when the profile is being used. To graphically emphasize some of the concepts defined in the profile, we introduce new icons. This is the case of the transformation and relation concepts.

STEREOTYPE Transformation

Extended metaclass: Class

Graphical Notation: a transformation occurrence is shown by an hexagonal figure containing the name of the transformation and the <<transformation>> stereotype

Constraints:

[1] A transformation should contain relations or queries and should have at least two models as parameters (one input and one output)

```
(self.relations()->notEmpty() or
self.queries() -> notEmpty()) and
self.modelParameters() -> size() > 1
```

[2] The model parameters set of the transformation must be equal to the domain and codomain type models set.

```
self.modelParameters()=self.relations
()-> collect (relation: Class |
```

[1] There is neither Dependency, nor any other Relationship in the Infrastructure connecting elements different from a Classifier.

```
relation. typedModels()) -> flatten()
-> asSet()
```

[3] A transformation can only be related to relations or typedModels

```
self.relatedElements()->forAll(re|
re.stereotype.name='relation' or
re.stereotype.name='typedModel')
```

Additional Operations

[1] The query relations() gives all the relations associated with the transformation

```
Class:: relations(): Set (Class)
relations =
self.owningPackage.ownedMember->
select ( r:Class | r.stereotype.name
= 'relation' and
self.owningPackage.ownedMember ->
exists (a:Association | a.memberEnd -
> includes (e: Property | e.type =
self and
e.opposite.type = r))
```

[3] The query modelParameters() gives all the typedModels associated with the transformation

```
Class:: modelParameters (): Set
(Package)
modelParameters =
self.owningPackage.ownedMember->
select ( m: Package |
m.stereotype.name = 'typedModel' and
self.owningPackage.ownedMember ->
exists
(d:DirectedRelationship | d.source =
self and d.target = m))
```

STEREOTYPE Relation

Extended metaclass: Class

Graphical Notation: a Relation occurrence is shown by a pentagonal figure containing the name of the transformation and the <<relation>> stereotype

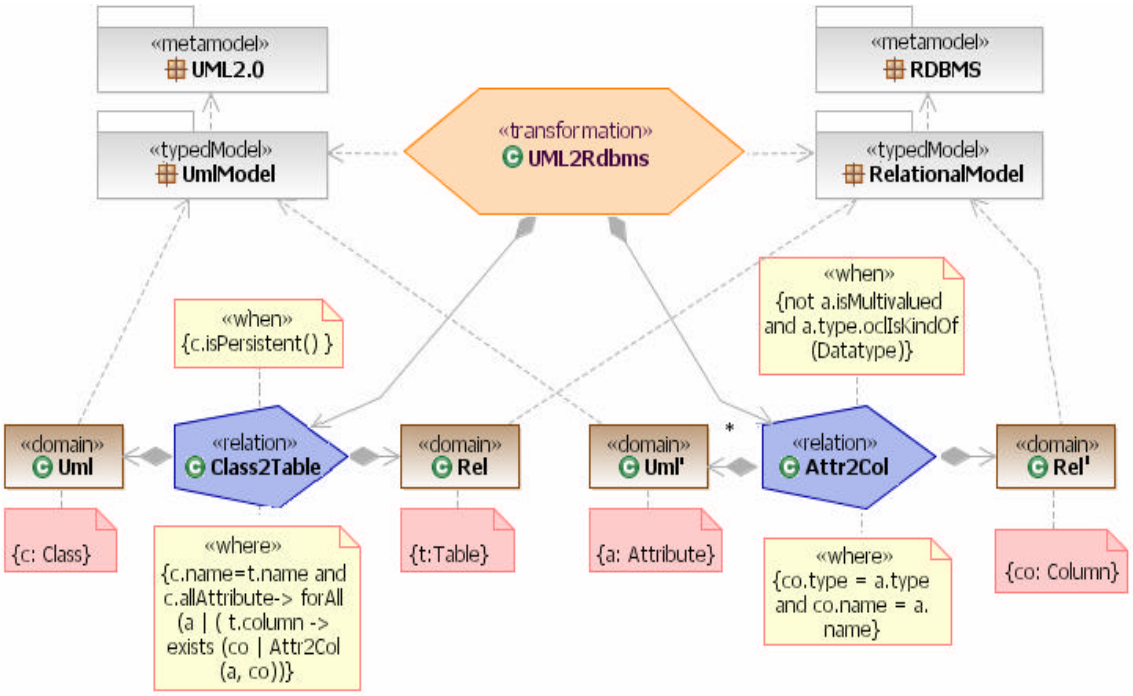


Figure 6. Instantiation of the profile for the example

[2] The query queries() gives all the queries associated with the transformation

```
Class:: queries(): Set (Operation)
queries = self.ownedOperations-
>select (o:Operation |
o.stereotype.name='query')
```

OwnedAttributes:

- isTopLevel ::Boolean - Specifies whether the relation is a top level relation. Default value is false. If isTopLevel is false, the relation must be explicitly called by another one.

Constraints:

[1]. A relation should be defined in at least one domain and one coDomain

```
self.domain()-> size()>=1 and
self.coDomain() -> size()>= 1
```

[2] A relation may have a when clause. However, it must have a where clause.

```
(self.constraint -> select(c |
c.stereotype.name = 'when')->
size()<= 1) and (self.constraint ->
select(c | c.stereotype.name 'where')
-> size()= 1)
```

Additional operations

[1] The query domain() gives all the domains associated with the relation through role name domain.

```
Class:: domain (): Set (Class)
domain =
self.owningPackage.ownedMember ->
select ( d:Class | d.stereotype.name
= 'domain' and
self.owningPackage.ownedMember ->
exists (a:Association |
a.memberEnd -> includes (e: Property
| e.type = self and e.opposite.type =
d and e.opposite.name = 'domain')))
```

[2] The query coDomain() gives all the domains associated with the relation through role name codomain.

```
Class:: coDomain(): Set (Class)
coDomain =
self.owningPackage.ownedMember->
select ( d:Class |
d.stereotype.name = 'domain' and
self.owningPackage.ownedMember ->
exists (a:Association |
a.memberEnd -> includes (e: Property
| e.type = self and e.opposite.type =
d and e.opposite.name = 'codomain')))
```

[3] The query transformation() gives all the transformation associated with the relation

```
Class:: transformation (): Set
(Class)
transformation =
self.owningPackage.ownedMember->
select ( t:Class | t.stereotype.name
= 'transformation' and
self.owningPackage.ownedMember ->
exists (a:Association | a.memberEnd -
> includes (e: Property | e.type =
self and e.opposite.type = t)))
```

[4] The query typedModels() gives all the type models associated with the relation (the domain and codomain typed model)

```
Class:: typedModels(): Set (Package)
typedModels =
self.domain().typedModel -> union
(self.coDomain().typedModel)
```

STEREOTYPE Domain

Extended metaclass: Class

OwnedAttributes:

- isChecked:: Boolean - Specifies whether the domain is just checked. In a checked domain, the constraints will be evaluated regarding the current model-associated values.
- isEnforced:: Boolean - Specifies whether the domain is forced. Default value is false. In a forced domain object creation and/or erasure may occur.

Constraints:

[1] A domain should make reference to only one TypedModel (there is a DirectedRelationship between Domain and TypedModel)

```
self.owningPackage.ownedMember->
select(d:DirectedRelationship |
d.source = self and
d.target.stereotype.name =
'typedModel')-> size()=1
```

[2] A Domain must contain at least one variable declaration

```
self.variables() -> size()>0
```

Additional operations

[1] The query relation() gives all the relation associated with the domain .

```
Class:: relation(): Set (Class)
relation =
self.owningPackage.ownedMember->
select ( d:Class | d.stereotype.name
= 'relation' and
self.owningPackage.ownedMember
-> exists ( a:Association |
a.memberEnd -> includes (e :
Property| e.type = self and
e.opposite.type = d))
```

[2] The query variables() gives all the variable declarations associated with the domain .

```
Class:: variables(): Set
(VariableDeclaration)
variables = self.constraint -> select
( c:Constraint |
c.body.bodyExpression.oclIsKindOf(Var
iableExp))-> collect (c:Constraint |
c.body.bodyExpression.referredVariabl
e)
```

STEREOTYPE TypedModel

Extended metaclass: Package

Constraints:

[1] A typed model should be an instance of one metamodel, that is, it should be the source of a DirectedRelationship having as target a <<metamodel>> package

```
self.metamodel() -> size() = 1
```

[2] A typed model should only contain instances of the metamodel it relates to.

```
self.ownedMember-> forAll (e|
self.metamodel().ownedMember() ->
includes (e.type))
```

Additional operations

[1] The query metamodel () gives all the metamodel packages associated with the typedModel .

```
Package:: metamodel (): Set (Package)
metamodel =
self.owningPackage.ownedMember
->select (d:DirectedRelationship |
d.source = self and
d.target.stereotype.name =
'metamodel') -> collect
(d:DirectedRelationship | d.target) )
```

STEREOTYPE When/Where (subclass of Invariant)

Extended metaclass: ExpressionInOCL

Constraints:

[1] A when/where clause must constraint a relation.

```
self.constraint.constrainedElement.
stereotype = 'relation'
```

5. Profile Application to the Example

In this Section, we show an instance of the metamodel using the proposed profile for the redefined example in Section 3. Figure 6 illustrates this example using the proposed graphic notation and stereotypes. We can see that a new instantiation of this transformation will be upon particular elements of concrete models at the M1 level of the 4-layer architecture. This model presents a transformation instantiation that defines transformation patterns upon the generic instances of the metaclasses (e.g. classes and tables, attributes and columns). The transformation has two models as parameters (umlModel and RelationalModel) and two relations (class2Table and Attr2Col). When and where clauses are OCL expressions stereotyped with <<when>> and <<where>> respectively, which constrain the relations Class2Table and Attr2Col. Both relations have one domain (Uml) and one codomain (Rel). Because of domains and codomains are structurally identical and the only difference between them are just its role into the relation, it is possible to represent both by a class stereotyped with <<domain>>. Variable declarations within the domains are also expressed in OCL via a VariableDeclaration metaclass that, as stated in subsection 4.2, matches the TemplateExp metaclass of the QVT metamodel.

6. Conclusions and Future Work

Model Transformation is the MDD engine; in this way, models, from being mere contemplative entities, become productive entities within the software development process. Model transformations require specific languages for their definition. These languages must have a formal base, for instance, a metamodel that supports them, and allows for an automated treatment.

In this paper, we have proposed a pure declarative language to express model transformations whose initial metamodel is inspired in the QVT language. Regarding the construction of a language, there are two options: One of them is create a new language based in a metamodel implicating new metaclasses definition. Another option is using the standard extension mechanism of UML that consists in new stereotypes definition extending existing metaclasses.

Our proposal is based on specifications already existing in OMG; on the one hand, by means of the stereotype definition, we extend Infrastructure 2.0, an independent and more abstract specification than UML; on the other hand we use the OCL language to express transformation patterns, since after analyzing these concepts, we concluded that it is not necessary to create new elements for modeling them.

The proposed language is expected to be minimal in the set of metamodels permitting us to express relations and queries of model transformation. To maintain simplicity and to reduce user training time are the main

advantages of this minimal approach. Furthermore, since the proposal maximizes the use of OCL and there are several modeling tools that support OCL, the development work of a tool to support this transformation language is much easier.

As future working lines, in the near future, we will implement this language integrating it to the Case tool [15] that our research group is developing. This tool, oriented to formal modeling, includes the model refinement specification which can be seen as a particular model transformation case. On this topic, we have published, among other, papers [16, 17] which provide a formal base to this tool. Some of the works about model refinement that have helped us as inspiration are [13, 14]. Furthermore, we propose ourselves to incorporate to the language elements to facilitate traceability among models, consistency checking (when the transformation is forced), testing integrated to transformation, and transformation composition.

7. References

[1] MDA Guide, v1.0.1, omg/03-06-01, June 2003.

http://www.omg.org

[2] OMG (Object Management Group) http://www.omg.org

[3] Meta Object Facility (MOF) 2.0. OMG Adopted Specification. October, 2003. http://www.omg.org

[4] MOF 2.0 Query/View/Transformations - OMG Adopted Specification. March 2005. http://www.omg.org

[5] OMG. The Object Constraint Language Specification – Version 2.0, for UML 2.0, revised by the OMG, http://www.omg.org. April 2004.

[6] The Unified Modeling Language Superstructure version 2.0.,OMG April 2004. http://www.omg.org

[7] The Unified Modeling Language Infrastructure version 2.0, OMG. March 2005. http://www.omg.org

[8] Jouault F., Kurtev I. Transforming Models with ATL Workshop in Model Transformation in Practice at the MODELS 2005 Conference. Montego Bay, Jamaica, 2005

[9] Akehurst D., Howells W., McDonald-Maier K. Kent Model Transformation Language. Workshop in Model Transformation in Practice at the MODELS 2005 Conference. Montego Bay, Jamaica, Oct 3, 2005

[10] Lawley M., Steel J. Practical Declarative Model Transformation with TefKat. Workshop in Model Transformation in Practice at the MODELS 2005 Conference. Montego Bay, Jamaica, Oct 3, 2005

[11] Dimitrios S. Kolovos, Richard F. Paige and Fiona A.C. Polack. The Epsilon Object Language (EOL). In Proc. Model Driven Architecture Foundations and Applications: Second European Conference, ECMDA-FA, volume 4066 of LNCS, pages 128– 142, Bilbao, Spain, June 2006.

[12] Anneke Kleppe. MCC: A Model Transformation Environment. A. Rensink and J. Warmer (Eds.): ECMDA-FA 2006, LNCS 4066, pp. 173 – 187, Spain, June 2006.

[13] Hnatkowska B., Huzar Z., Tuzinkiewicz L. On Understanding of Refinement Relationship. Workshop in Consistency Problems in UML-based SoftwareDevelopment III – Understanding and Usage of Dependency Relationships at the 7th International UML Conference.. Portugal, 2004.

[14] Liu Z., Jifeng H., Li X., Chen Y. Consistency and Refinement of UML Models. Workshop in Consistency Problems in UML-based Software Development III at the 7th International Conference on the UML.Portugal, 2004

[15] Pons C., Giandini R., Pérez G., et al. Precise Assistant for the Modeling Process in an Environment with Refinement Orientation. In UML 2004 Satellite Activities, Revised Selected Papers". Lecture Notes in Computer Science number 3297. Springer, Oct., 2004.

[16] Pons, C., Kutsche, R. Traceability Across Refinement Steps in UML Modeling. 3rd WiSME at the 7th UML Conference. October 11, 2004. Lisbon, Portugal .

[17] Pons, C., Pérez,G., Giandini, R., Kutsche, R. Understanding Refinement and Specialization in the UML. 2nd International Workshop on Managing SPEcialization/Generalization Hierarchies (MASPEGHI). In IEEE ASE 2003, Canada.

[18] Giandini, R., Pons, C. Un lenguaje para Transformación de Modelos basado en MOF y OCL. Proceedings of XXXII CLEI. Santiago, Chile. August, 2006.

Extensión MDA (Model Driven Architecture) para proceso basado en RUP (Rational Unified Process)

Andrea Delgado, Natacha Carballal, Catalina Rapetti
Universidad de la República, Facultad de Ingeniería, Instituto de Computación
{adelgado, pgmdapis}@fing.edu.uy

Resumen

El enfoque de desarrollo Model Driven Architecture (MDA) propone basar el desarrollo en modelos, separando la especificación del sistema de las plataformas utilizadas. El Grupo de Ingeniería de Software (Gris) del Instituto de Computación tiene como eje de sus actividades un programa de construcción y prueba de modelos de proceso en el curso "Proyecto de Ingeniería de Software" desde el año 2000. El objetivo de este trabajo en progreso es realizar y probar una Extensión MDA al proceso basado en RUP que se tiene definido, modificando principalmente las actividades y entregables asociados a las Disciplinas de Diseño e Implementación, y seleccionando una herramienta con soporte MDA. Se presenta el enfoque MDA, el proceso basado en RUP y el contexto y prueba de la Extensión MDA definida.

1. Introducción

El grupo de Ingeniería de Software (Gris) del Instituto de Computación de la Facultad de Ingeniería de la Universidad de la República, Uruguay [1] tiene como eje principal de sus actividades un programa de construcción y prueba de modelos de proceso, que inició en el año 2000 y utiliza como banco de pruebas de los procesos al curso "Proyecto de Ingeniería de Software" [2] que se dicta en cuarto año de la carrera de Ingeniería en Computación.

El objetivo central del programa es el desarrollo de modelos de proceso que puedan resultar adecuados para su transferencia a la industria, para lo que, entre otras características, se cuenta con clientes externos al curso. Estos clientes plantean el desarrollo de sistemas de mediano porte que en general presentan desafíos tecnológicos importantes y son en general grupos de investigación, proyectos o áreas de la misma facultad, o empresas del medio. Desde su puesta en marcha y hasta la fecha se han estudiado y adaptado varios modelos de proceso de las distintas corrientes existentes, tanto procesos "pesados" como "ágiles", siendo el representante más importante de los primeros el Rational Unified Process (RUP) [3] y de los segundos el eXtreme Programming (XP) [4], si bien existen "agilizaciones" del RUP que no se han abordado. De los dos procesos mencionados se han realizado adaptaciones que se han puesto a prueba en distintas ediciones del curso, definiéndose como proceso base del programa la adaptación realizada del RUP que es sobre la cual se agregan otros enfoques y metodologías de desarrollo cuya descripción puede verse en [5].

La metodología para el desarrollo de aplicaciones con enfoque MDA se inscribe en este programa de prueba de procesos y está siendo desarrollada por parte de un proyecto de fin de carrera de quinto año de la carrera de Ingeniería en Computación, dirigido por un docente del Grupo de Ingeniería de Software (Gris). La definición de la Extensión MDA [6] se basa en el estándar del Object Management Group (OMG) [7], agregando actividades y entregables específicos al proceso base adaptación del RUP, y la utilización de una herramienta que cumpla con el enfoque MDA y que mejor se adapte a las características del curso.

2. Model Driven Architecture (MDA)

El enfoque de desarrollo Model Driven Architecture (MDA) según la especificación en [7] se basa en la utilización de modelos en el desarrollo de software, y en la idea de separar la especificación de un sistema, de los detalles de la forma en que dicho sistema utiliza las capacidades de sus plataformas. Está guiado por modelos en tanto provee sentido a la utilización de modelos como base para dirigir el curso de la comprensión, diseño, construcción, deployment, operación, mantenimiento y modificación de los sistemas. A partir de la separación de intereses planteada en el enfoque, se buscan obtener tres metas principales para los sistemas desarrollados: portabilidad, interoperabilidad y reusabilidad. Esta separación de intereses se ve reflejada en la definición de las tres vistas de un sistema que se proveen en los modelos definidos: Computation Independent Model (CIM), Platform Independent Model (PIM) y Platform Specific Model (PSM). El ciclo de vida en un desarrollo utilizando MDA comienza como en

cualquier proceso de desarrollo por la captura de requerimientos. Estos en su mayoría se especifican en forma de texto, sobre los cuales se realiza el análisis de requerimientos que da lugar al primer modelo del sistema que es un modelo conceptual del mismo. Este modelo en el enfoque MDA se conoce como CIM (Computation Independent Model) para el cual en general no se definen transformaciones. A partir del CIM en forma manual se realiza el PIM (Platform Independent Model) que es un modelo de diseño independiente de la implementación, sobre el cual se definen transformaciones que llevarán ese modelo al PSM (Platform Specific Model) a partir del cual se generará el código correspondiente a la plataforma elegida. Es importante notar que a partir de un mismo PIM se pueden generar tantos PSM como plataformas objetivo se definan, y también se puede generar directamente el código sin pasar por un PSM. El aspecto central del enfoque se encuentra en la transformación de modelos, que según la definición provista por el estándar, es el proceso de convertir un modelo en otro modelo del mismo sistema. En la figura 1 se presenta la secuencia general de pasos definidos en la aplicación del enfoque según [8]:

Figura 1. Los tres pasos principales del enfoque MDA

2.1. Conceptos en MDA

En esta sección se describen los principales conceptos que definen el enfoque MDA según [7].

El Computation Independent Model (CIM) es una vista del sistema desde el punto de vista independiente de la computación. Puede estar compuesto por varios artefactos como Modelo de Dominio, especificación de requerimientos como Modelo de Casos de Uso, y es independiente de cómo el sistema es implementado. Actúa como entrada para el resto de los modelos y como fuente de vocabulario compartido para usar en otros modelos. Los requerimientos establecidos en el CIM deben ser trazables al PIM y al PSM que los implementan, y viceversa.

El Platform Independent Model (PIM) es una vista del sistema desde el punto de vista del diseño independiente de la plataforma. Describe el sistema desde el punto de vista del software que lo compone, sin presentar detalles de cómo serán utilizadas las plataformas asociadas. En base a dicha independencia se puede utilizar en la generación de modelos específicos para varias plataformas distintas. Por ejemplo, se puede asumir la disponibilidad de características de tipo general de plataforma, como ser invocación remota, sin especificar más que eso, luego la característica indicada será implementada según la plataforma.

El Platform Specific Model (PSM) es una vista del sistema desde el punto de vista específico de la plataforma. Un PSM combina las especificaciones en el PIM, con los detalles que definen como el sistema utiliza ese tipo particular de plataforma. El PSM puede ser el código para la plataforma elegida, si provee toda la información necesaria para construir el sistema y ponerlo en operación, o puede actuar como un PIM que se utilizará para refinamientos posteriores en un PSM que pueda ser implementado directamente.

Los Mapeos entre modelos proveen las especificaciones para realizar las transformaciones de un PIM en un PSM para una plataforma específica. El modelo asociado a la plataforma, determinará la naturaleza de los mapeos. Por ejemplo, un PIM especificado en UML mapeado a EJB provee marcas que serán utilizadas para guiar la transformación del PIM al PSM, de forma que marcando una clase con la marca de “session”, resulta en la transformación de esa clase según el mapeo establecido, en un “session bean” y otras clases soporte. Un mapeo es especificado utilizando algún lenguaje para describir una transformación de un modelo en otro. La descripción puede ser en lenguaje natural, un algoritmo en un lenguaje de acciones según lo establecido en el Precise Action Semantics incorporado a UML 2.0, o en un lenguaje de mapeo de modelos como se define en el estándar QVT. Para realizar el mapeo de modelos se definen dos enfoques: mapeo de tipos y de instancias.

El Marcado de modelos se utiliza para los mapeos, dichas marcas provienen de distintas fuentes: tipos de un modelo (clases, asociaciones, etc.), roles de un modelo (patrones), estereotipos de un perfil UML, elementos de un modelo MOF, elementos de modelos especificados por cualquier metamodelo. Para que las marcas puedan ser utilizadas adecuadamente, deben ser estructuradas, restringidas y/o modeladas, por ejemplo un conjunto de marcas que indique alternativas mutuamente excluyentes para un concepto, deben ser agrupadas de forma que al marcar el modelo se sepa cuales son las posibles elecciones y que más de una de esas marcas no pueden ser aplicadas al mismo elemento del modelo. Algunas marcas incluso podrían necesitar parámetros, como las de QoS, por ejemplo “soportar conexiones simultáneas” podría requerir un parámetro que indique cota superior de la cantidad de conexiones que debe soportar, o incluso varios parámetros indicando detalles para time-outs o políticas de conexión. Un conjunto de marcas podría ser especificado por un modelo de marcas independiente de los mapeos, en vez de ser provisto por un mapeo en particular, y así ser usado por distintos mapeos, o ser provisto con un perfil UML.

La Transformación entre modelos es la base del enfoque, luego que se tiene el PIM marcado, el

siguiente paso es transformarlo en un PSM. Esta transformación puede ser realizada según distintos enfoques: en forma manual, asistida por computadora o en forma automática. La entrada para la transformación es el PIM marcado y el mapeo, el resultado de la misma es el PSM obtenido o el código generado y el registro de la transformación. La transformación es realizada según el tipo de mapeo que se utilice: de tipos o instancias en modelos. Los resultados de la transformación de un PIM utilizando una técnica particular, son el PSM obtenido en la transformación y el registro de la misma. Este registro contendrá un mapeo del elemento en el PIM a los correspondientes elementos en el PSM, mostrando que parte del mapeo fue utilizado en cada parte de la transformación. Una herramienta que lleve un registro de la transformación podría sincronizar el PIM y el PSM generado, cuando alguno sea modificado, y con el código obtenido.

2.2. Modelado y estándares en OMG

El Object Management Group (OMG) utiliza una arquitectura de cuatro capas de modelado sobre la cual está basada la definición de sus estándares, y el concepto de metamodelado definido como: el uso de modelos (en una capa de abstracción) para describir modelos (en una capa inferior siguiente). Estas capas se notan M0, M1, M2 y M3; cada una y sus dependencias están definidas por el OMG y especificadas en [7] como sigue:

Capa M3, el Meta-Meta-Modelo: contiene entidades para definir lenguajes de modelado. El estándar Meta-Object Facility (MOF) que se ubica en esta capa, es el lenguaje para definir lenguajes de modelado, por ejemplo una "MOF Class". Los elementos de la capa M3 se definen con instancias de conceptos de esta capa, o sea que MOF se define en base a MOF.

Capa M2, el Meta-Modelo: en esta capa se ubican los metamodelos que contienen las entidades de los lenguajes de modelado definidos. Los estándares Unified Modeling Language (UML) y Common Warehouse Metamodel (CWM) están en esta capa, y definen los elementos con los cuales modelar sistemas, por ejemplo "UML Class". Los elementos en esta capa son instancias de los elementos de la capa M3.

Capa M1, el Modelo: contiene las entidades con que se modela un sistema en particular, según el lenguaje de modelado utilizado, por ejemplo en UML podría ser la Clase Socio con Atributos Nombre y Apellido. Los elementos en esta capa son instancias de los elementos en la capa M2.

Capa M0, los Datos: contiene los datos específicos que maneja el sistema modelado según la capa M1, por ejemplo el Socio "Juan Perez", que es una instancia de la Clase Socio con Atributos Nombre y Apellido definidos en la capa M1. Los elementos en esta capa son instancias de los elementos en la capa M1.

Además define una variedad de estándares sobre algunos de los cuales se basa el enfoque MDA. La importancia de los mismos radica en que la utilización de estos estándares permite realizar en forma automática, validación y ejecución de modelos, y transformación de modelos, lo que como ya se mencionó, constituye el elemento central del enfoque.

Meta Object Facility (MOF): establece un lenguaje común para definir lenguajes de modelado como UML, lo que permitirá definir transformaciones sobre los metamodelos en forma estándar. En base a MOF se define un repositorio de modelos que puede utilizarse para especificar y manipular modelos.

XML Metadata Interchange (XMI): permite el intercambio de modelos via documentos XML. XMI se utiliza para intercambiar modelos en el nivel M1, y para generar formatos de intercambio de metamodelos en la capa M2.

Unified Modeling Language (UML): lenguaje de modelado estándar para especificar, visualizar y documentar sistemas de software. En UML 2 se integran un conjunto de conceptos para especificar completamente el comportamiento de objetos, el UML Precise Action Semantics, que permite agregar dicha información.

Perfil UML: mecanismo de extensión de UML. Se aplica a la especificación de un lenguaje, especificando un nuevo lenguaje de modelado mediante el agregado de nuevos elementos o restricciones al lenguaje. Varios perfiles pueden aplicarse a un modelo, extendiendo o restringiendo los elementos de ese modelo. Actualmente existen perfiles para CORBA, Java, EJB y C++, lo que permitirá construir PSMs específicos para estas tecnologías.

Object Constraint Language (OCL): lenguaje de especificación para escribir expresiones sobre modelos, por ejemplo invariantes, pre y postcondiciones, y cuerpo de operaciones. Mediante OCL se podrían especificar transformaciones describiendo los elementos en el modelo origen y destino.

Query, Views and Transformations (QVT): define lenguajes usando MOF para especificar como se realizan las transformaciones entre modelos. Es un estándar con gran relevancia para MDA, ya que establece un modo estándar de definir transformaciones entre modelos. Se definen lenguajes para crear vistas de un modelo, realizar consultas sobre modelos y escribir definiciones de transformaciones entre modelos, siendo este último el de mayor relevancia para MDA, el Model Transformation Language (MLT) que utiliza el "pattern matching" como uno de sus factores clave para permitir la definición de transformaciones entre modelos.

3. Proceso y metodología de desarrollo

Muchos procesos de desarrollo de software en la actualidad tienen en cuenta que los requerimientos de los sistemas son inestables y debe ser posible incorporar cambios en los mismos durante el desarrollo a medida que van surgiendo, así como agregar nuevos requerimientos. Estos modelos, ya sean "pesados" o "ágiles", siguen la filosofía de la interacción y el cambio, e indican liberaciones frecuentes que son utilizadas para obtener feedback de los usuarios, y la construcción del sistema en forma iterativa incremental sobre estas liberaciones.

Una metodología para el desarrollo basado en MDA incorpora la concepción de que los modelos que se realizan en las distintas etapas, no son meramente documentación del sistema para utilizar durante las distintas etapas del desarrollo y posterior mantenimiento de la aplicación, sino que constituyen la aplicación, siendo en base a estos que se realizan los cambios necesarios y a partir de estos se regenera el código que constituye el sistema operacional. Para esto se deben incorporar actividades, roles y entregables específicos para el enfoque, así como una herramienta de desarrollo que lo soporte.

3.1. Proceso de desarrollo basado en el RUP

El principal proceso de desarrollo base con que cuenta la Organización, en este caso el Grupo de Ingeniería de Software (Gris) en el curso "Proyecto de Ingeniería de Software", es una adaptación del Rational Unified Process (RUP) que ha ido evolucionando desde su primera versión que fuera definida por estudiantes en el año 2000 como proyecto de grado [9]. A partir del año 2004 se compone de un esqueleto de actividades que es común a todos los tipos de desarrollo que se emprendan, y dos extensiones principales para realizar desarrollos específicos en OO (Orientación a Objetos) y Genexus [10], herramienta de cuarta generación para el desarrollo de aplicaciones orientadas a bases de datos. En este contexto la metodología de desarrollo para aplicaciones con enfoque MDA se incorpora al esqueleto base más la extensión OO, agregando actividades, roles y entregables específicos, así como una herramienta adecuada para el desarrollo con dicho enfoque.

El proceso base tiene como el RUP dos dimensiones, el tiempo y las disciplinas. En la dimensión del tiempo se definen cuatro fases: Inicial, Elaboración, Construcción y Transición, en las tres primeras fases se definen dos iteraciones de dos semanas cada una, y en la última fase una iteración de dos semanas, que completan junto con la semana de preparación previa y la semana de evaluación final, las dieciséis semanas con que cuenta el curso. Se cuenta además con una agenda definida que indica para cada semana que actividades se deben realizar y que entregables se deben obtener. En la dimensión de las Disciplinas se definen las disciplinas tradicionales del desarrollo de software: requerimientos, diseño, implementación y verificación, más las disciplinas de soporte: gestión del proyecto, de la calidad, de la configuración e implantación. En cada disciplina se definen actividades, roles encargados y participantes de las actividades, y entregables de entrada y salida de dichas actividades. Una descripción completa del proceso base incluyendo su concepción y evolución puede consultarse en [5].

3.2. Roles, Actividades y Entregables en MDA

En la definición del Proceso Extensión MDA, no se hicieron cambios en los roles ni a las combinaciones de roles definidos en el proceso base, siendo los participantes más importantes el Arquitecto, los analistas y los Especialistas Técnicos. Se vió que era necesario contar con algún asistente de Arquitecto, ya que éste se veía sobrecargado de trabajo. También se detecto la necesidad de que uno de los Especialistas Técnicos se dedicara solamente a la herramienta MDA, ya que el aprendizaje de la misma, requiere dedicación y pruebas. Como hay que modelar de forma tal que la herramienta MDA pueda traducir el modelo en código, es necesario aprender para que sirve cada estereotipo y tagged value por tecnología. Los perfiles UML son los que permiten agregar esta información adicional al modelo y que la herramienta interprete como generar el código, dependiendo de las marcas que se le asignen a los diferentes objetos del modelo. Esta es la tarea principal del Especialista MDA, quien luego transmite su conocimiento al resto de los integrantes del equipo.

La aplicación del enfoque MDA comienza desde la captura de requerimientos en el CIM, el diseño del PIM en la herramienta seleccionada, y la generación del PSM y/o código basado en las tecnologías a utilizar. En la disciplina de Requerimientos no se agregan actividades sino que se conceptualiza el enfoque agrupando actividades que ya existen. En la disciplina de Diseño se agregan las actividades más importantes del enfoque que tienen que ver con el modelado del PIM y el marcado del mismo para poder generar el código correspondiente, tarea que se realiza en la disciplina de Implementación. Además, para cada actividad se definen los roles responsables, participantes y sus entregables de entrada y salida.

3.2.1. Disciplina Requerimientos: En esta disciplina participan principalmente los roles de Analista y Arquitecto, quienes realizan principalmente el relevamiento de requerimientos a partir de reuniones

con el cliente. Para el enfoque MDA no se realizan agregados de actividades sino que se define la conceptualización del CIM. Este modelo, generalmente se hace también en el proceso base, pero consta de tres partes: la especificación de requerimientos, el modelo de casos de uso y el modelo de dominio.

Especificar el CIM (R11): tiene como objetivo la conceptualización del CIM y consta de la reunión de los tres documentos como el modelo mencionado. El rol responsable de este entregable es el analista.

3.2.2. Disciplina Diseño: En la disciplina de Diseño, participan los roles de Analista, Arquitecto y los Especialistas técnicos, donde los primeros aportan su conocimiento de los requerimientos del cliente, el Arquitecto traduce los requerimientos en el diseño de la solución al problema planteado, mientras los últimos aportan el conocimiento técnico de las herramientas. El enfoque MDA agrega dos nuevas actividades que se realizan además de las ya definidas en el proceso base, que tienen como rol principal al arquitecto, ayudado por analistas y especialistas técnicos.

Especificar el PIM (D6): consiste en crear un modelo del sistema en alto nivel. También se compone de salidas de actividades definidas en el proceso base, como la Descripción de la Arquitectura, el modelo de diseño, diagramas de Subsistemas, Clases, Colaboración, Secuencia, y el modelo de datos. El diagrama de clases se hace en UML, que luego es marcado con estereotipos y tagged values para conformar el PIM marcado que es el resultado de la siguiente actividad.

Definir Marcas para el PIM (D7): como ya se menciono, se agrega información adicional al modelo en UML, para que este pueda ser transformado automáticamente, dependiendo de la tecnología de desarrollo a utilizar. Junto con el PIM marcado se entrega una planilla de marcas, que tiene como objetivo facilitar la tarea de estereotipado, y consta de un cuadro que indica a que clase le corresponde que estereotipo y quien es el encargado de dicha clase.

3.2.3. Disciplina Implementación: En esta disciplina se encuentran definidas actividades para realizar la implementación del diseño definido, y los roles participantes son Especialistas Técnicos e implementadores. Para el enfoque MDA se modifican actividades y se agregan nuevas.

Investigar la herramienta de desarrollo (I8): esta actividad es del proceso base y se le agrega el estudio de la o las herramientas que se utilizarán para MDA.

Crear las clases y subsistemas (I10, I11): son actividades del proceso base que se ven modificadas pues el trabajo de generación de código manual se espera sea mucho menor, ya que la mayoría del código es generado automáticamente por la herramienta MDA.

Especificar el PSM (I13): consiste en transformar el PIM marcado obtenido anteriormente, para obtener un modelo especifico de la plataforma a utilizar, en caso que se utilice más de una plataforma será uno por cada plataforma elegida.

Definir características del proyecto (I14): el objetivo de esta actividad es configurar el ambiente de desarrollo, con la herramienta MDA a utilizar en el proyecto. Esta información se almacena en una planilla de forma que la misma sea un referente para el equipo a la hora de trabajar con la herramienta MDA.

3.3. Selección de herramientas MDA

Para poder extender el proceso con el enfoque MDA, además de definir las actividades y entregables, se debía proporcionar la herramienta MDA. Para ello se investigaron algunas de las herramientas existentes en el mercado, buscando una que además de cumplir con el enfoque, fuera factible su uso en el curso "Proyecto de Ingeniería de Software" que plantea algunas restricciones como el uso de Eclipse [11] como IDE para el desarrollo. La selección consto de tres etapas: investigación de herramientas existentes, preselección de herramientas, y selección de la herramienta MDA a utilizar en el curso. Para definir una metodología de evaluación de herramientas, se utilizo como referencia el estudio en [12].

3.3.1. Investigación de herramientas MDA: Como salida de la investigación de herramientas se obtuvo una planilla de herramientas disponibles y sus características según la evaluación realizada, que constituyó la base para la siguiente etapa. Algunas de las características que se consideraron mas importantes para esta etapa se muestran en la Tabla 1:

Propiedad	Descripción
Licenciamiento	Free o si es comercial período de uso
Documentación	Existencia y calidad de la documentación
Plug-in Eclipse	Si es plug-in para Eclipse y de que versión
Reqs. de CPU disco, memoria	Requerimientos para instalación y ejecución
Generación de código	Tecnologías para las que genera código
Herramienta de modelado	Si incluye o no herramienta de modelado
Estándar UML	Versión UML 1.4 y/o 2.0
Estándar XMI	Versión XMI 1.x y/o 2.x

Tabla 1: criterios para evaluación de herramientas MDA

Las herramientas MDA evaluadas fueron: ArcStyler 5.5 Architect Edition, OptimalJ Professional Edition 4.1, IBM Rational Software Architect 6.0, AndroMDA 3.2, AndroIDE 0.0.5, Acceleo 1.0.1, TaylorMDA 0.0.2 y OpenMDX 1.12.1. La evaluación de características para algunas de las herramientas mencionadas se muestra en la tabla 2:

Propiedad	IBM RSA 6.0	AndroMDA 3.2
Licenciamiento	Free por 30 días.	Open Source
Documentación	Excelente+ videos explicativos	Buena
Plug-in Eclipse	Si	No
Reqs. de CPU disco, memoria	6 Gb disco, y > 512 Mb RAM	400 Mb de disco aprox.
Generación de código	C#, Java y EJB	Java, .NET y generar nuevos cartuchos
Herramienta de modelado	Trae incorporada en la herramienta MDA	No tiene, se debe combinar con alguna
Estándar UML	UML 2.0	UML 1.4 y 2.0
Estándar XMI	XMI 2.0	XMI 1.2

Tabla 2: evaluación de propiedades para herramientas MDA

3.3.2. Preselección de herramientas MDA: de la primera evaluación, se eligieron dos herramientas: una herramienta comercial Rational Software Architect de IBM [13] para la cual se cuenta con licencia universitaria, y una herramienta Open Source AndroMDA[14], que debía ser usada conjuntamente con una herramienta de modelado compatible ya que como la mayoría de las herramientas MDA que son open source, no es completa. Para poder utilizarla fue necesario estudiar que herramientas de modelado en UML hay disponibles, y cuales eran compatibles y factibles de usar según los requerimientos del curso. En la tabla 3 se muestra la evaluación de las principales características para algunas de las herramientas de modelado compatibles con AndroMDA:

Propiedad	MagicDraw 9.5	Poseidon 3.2.1	Eclipse UML
Licenciamiento	CE-free PE-licencia	CE-free PE-licencia	20 dias, licencia academica
Plataforma	Windows /Linux	Windows / Linux	Windows / Linux
Documentación	Buena	Buena	Buena
Plug-in Eclipse	No	No	Si
Estandar UML	1.4	1.4	1.4 y 2.0
Estandar XMI	1.2	1.2	2.0
Exportar XMI	Si	Si	No
Importar XMI	Si	Si	No

Tabla 3: evaluación de propiedades para herramientas de modelado

En un principio, se selecciono una herramienta de modelado que es plug-in de Eclipse pero no es gratuita: EclipseUML Studio Edition de Omondo[15]. Se obtuvo una licencia universitaria que permitía utilizarla en el curso. Esta herramienta ofrece la posibilidad de importar y exportar modelos, característica que otras herramientas no permitían o solo permitían exportar un archivo de imagen con el modelo, en lugar del modelo en si. Sin embargo, en la página de AndroMDA se indica que la herramienta es compatible pero puede tener problemas con la versión de XMI, por lo que se selecciono la segunda en la lista de recomendadas AndroMDA: Poseidon versión 3.2.1 [16] que posee una versión free, pero no es plugin de eclipse. Actualmente AndroMDA es compatible con versiones mas nuevas de las herramientas de modelado.

3.3.3. Selección de herramientas MDA: finalmente se debió decidir entre las dos herramientas preseleccionadas: Rational Software Architect y AndroMDA con Poseidón para el modelado. Teniendo en cuenta que la utilización de RSA implicaba realizar todo el desarrollo, incluyendo el modelado, en la misma herramienta, y dado que consume muchos recursos y las maquinas con que cuenta la facultad, no eran capaces de soportarla, se optó por utilizar la segunda opción, AndroMDA, que solo requiere la instalación de maven y un plugin, utilizando una consola para generar el código, que puede ser editado utilizando Eclipse. Además es desarrollada por la comunidad, por lo que creemos que tiene valor su utilización para generar aportes a dicho desarrollo.

Sin embargo, durante el desarrollo de la Fase Inicial se encontraron problemas con la herramienta de modelado elegida, porque no era posible representar todo lo necesario en los modelos y por algunas incompatibilidades y falta de soporte en AndroMDA. Al consultar en el foro de AndroMDA, la sugerencia fue utilizar Magic Draw [17], pues es la usada por sus desarrolladores, por lo que es posible obtener soporte rápidamente. Esta herramienta se había descartado en un principio porque la licencia académica parecía no ser gratis y la edición CE tenía limitaciones con respecto al número de diagramas que se podían crear. Se realizaron entonces consultas con la empresa a fin de obtener una licencia académica, que fue concedida completamente gratis. Debido a ello, se decidió cambiar a dicha herramienta de modelado. La suite completa de herramientas seleccionadas para el proceso MDA definido se muestra en la figura 2:

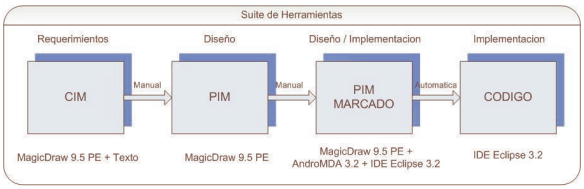


Figura 2. Suite de herramientas en el proceso MDA

4. Aplicación de la Extensión MDA

Para realizar la prueba de la Extensión MDA definida en el curso Proyecto de Ingeniería de Software correspondiente al segundo semestre del 2006, se eligió un proyecto que tuviera características que permitieran probar las capacidades del enfoque y de las herramientas seleccionadas. El Proyecto LIMS (Laboratory Information Management System) llevado adelante por el Instituto de Computación y el Instituto Pasteur de Montevideo IPMONT-FING [18] fue elegido debido principalmente a que planteaba varios desafíos en las tecnologías a utilizar y la reutilización de código existente ya desarrollado en otros proyectos.

4.1. Descripción del proyecto

Los LIMS son Sistemas para gestión de información en laboratorios, registran la información generada en los procesos y experimentos, y permiten manipular la información integrada facilitando: acceso compartido a resultados, según permisos de acceso que se otorguen, e implementación de mecanismos de colaboración entre investigadores y/o grupos. Si bien existen productos comerciales que proveen varias de estas funcionalidades, resultan poco adaptables a requerimientos específicos, por lo que surgen productos con fuerte componente académica por la necesidad de contar con herramientas especializadas.

El proyecto plantea la construcción de un LIMS según los requerimientos de IPMONT, con utilización de tecnologías abiertas como J2EE [19], servidores con Linux[20], servidor de aplicaciones Jboss[21] integrado con Jboss Portal[22] y JBPM[23], base de datos Postgress[24], y basado en estándares y otros proyectos como el EBI-PIMS[25]. El diseño de la aplicación tendrá fuerte orientación a servicios (Service Oriented Architecture SOA) con interfaces normalizadas, permitiendo la definición de módulos que sean integrados en forma incremental, y acceso Web que permita aumentar opciones de acceso y usabilidad y procesamiento distribuido, así como colaboración multi-institucional. En cuanto al proyecto específico definido para el curso Proyecto de Ingeniería de Software, los requerimientos incluyen funcionalidades de workflow de experimentos, movilidad de muestras en cada laboratorio y entre laboratorios, y gestión de proyectos que incluyen la realización de varios experimentos. Para el manejo de los elementos del dominio se reutilizará código de la herramienta desarrollada en el marco del EBI-PIMS, reutilizando el conocimiento biológico existente en dicho proyecto, como ser definición de samples y targets de experimentos.

4.2. Prueba de la Extensión MDA

Para realizar la prueba de la Extensión MDA se asignó dicho proyecto a dos grupos de estudiantes identificados como grupos 1 y 2, como forma de tener dos ocurrencias distintas del mismo proceso de desarrollo con el mismo proyecto, y poder evaluar los resultados obtenidos a la luz también de las comparaciones posibles entre los dos grupos y el desarrollo del mismo según las características de cada uno. Como primer tarea se presentó el proceso MDA definido, para lo cual se les brindó a los grupos la base conceptual del enfoque y de las actividades, entregables, roles y responsabilidades definidas. A medida que los grupos fueron avanzando en los requerimientos y modelado en la herramienta seleccionada, se realizó soporte de la misma en cuanto a aspectos relacionados con la realización del PIM y del PIM marcado, entrada para la generación con AndroMDA. Como se mencionó, al final de la Fase Inicial fue necesario que cambiar la herramienta de modelado ya que se detectaron algunas incompatibilidades y no era posible obtener soporte desde AndroMDA para la misma. Esta decisión creemos que fue adecuada para el avance de los proyectos, ya que a partir del cambio a MagicDraw pudieron generar el código en AndroMDA según las necesidades del proyecto.

La herramienta AndroMDA genera el código basado en el cartucho elegido, el cual provee los estereotipos necesarios para marcar las distintas clases e interpretar lo que se espera obtener. En la figura 3 se presenta un ejemplo de PIM marcado realizado por uno de los grupos que está siguiendo el proceso MDA:

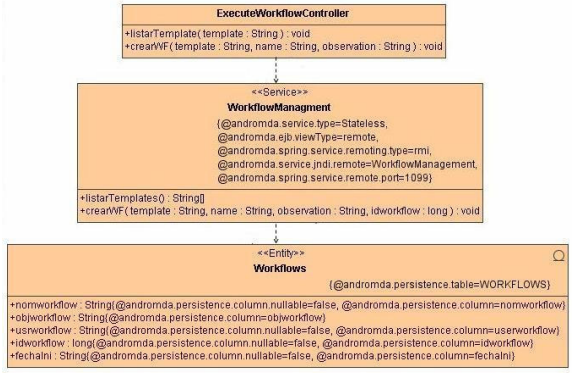


Figura 3. Ejemplo de PIM marcado en Magic Draw

Entre los cartuchos utilizados por los grupos para el proyecto se encuentran el JSF[26] para la presentación, Spring[27] y J2EE para la lógica, JBPM para la generación de workflows, y Postgress para la persistencia. La generación provee la aplicación de patrones de diseño como ser Data Access Object(DAO) con Hibernate[28] para el acceso a datos, obteniendose en este caso una generación del 100% del código incluyendo las clases para el mapeo entre paradigmas OO y relacional, y los scripts para la generación del esquema de la BD. Para otros requerimientos como ser la generación de portlets para el Jboss Portal no hay aún cartucho que permita realizarla desde la herramienta, por lo que este aspecto se tuvo que implementar sin generación.

4.3. Seguimiento de la prueba Extensión MDA

Para el seguimiento de la prueba de la Extensión MDA definida, según lo establecido en el proceso base, se cuenta con dos actividades principales: la revisión semanal con el Director de Proyecto que en este caso es quién también dirige el proyecto de grado de los estudiantes que definieron la Extensión MDA, y Auditorías a los grupos por parte de los docentes del curso, en este caso, conjuntamente con los estudiantes que definieron el proceso. Una descripción detallada de estas actividades puede encontrarse en [5].

Se realizaron hasta el momento dos Auditorías con los grupos, una al finalizar la Fase Inicial y otra al finalizar la Fase de Elaboración. Ambos grupos se atrasaron en el cumplimiento de los objetivos de la Fase Inicial, debido principalmente a aspectos relacionados con las tecnologías a utilizar, incluida la primer herramienta de modelado definida para MDA. En Fase de Elaboración se detectaron problemas en cuanto al procedimiento seguido para el manejo del modelo correspondiente al PIM marcado, ya que un grupo decidió que varias personas realizaran esta actividad a la vez, para lo cual cada uno se debía llevar una copia del modelo exportado en formato XMI, definiendo sobre que package trabajaría cada uno. Al momento de la integración se encontraron dificultades para volver a obtener un único modelo, lo que demandó más esfuerzo de integración del esperado.

A esta altura del proyecto los grupos han validado con el cliente el ejecutable de la Línea Base de la Arquitectura obtenido en la Fase de Elaboración, y que implementa los Casos de Uso relevantes a la Arquitectura. A pesar de las dificultades encontradas, ambos grupos han podido generar un alto porcentaje del código requerido para la aplicación en desarrollo, luego del cambio en la herramienta de modelado a Magic Draw, que se integra perfectamente con la herramienta de generación AndroMDA. El cliente ha validado el cumplimiento de los requerimientos básicos para la aplicación por parte de ambos grupos y las soluciones encontradas. Actualmente los proyectos están en Fase de Construcción, incorporando el resto de las funcionalidades incluidas en el Alcance para obtener el ejecutable final del proyecto.

5. Conclusiones y trabajo futuro

La Extensión MDA propuesta incorpora a las actividades, entregables y roles establecidos por el proceso base adaptación del RUP, la conceptualización del enfoque que se ve reflejada en las modificaciones y agregados definidos. Los aspectos clave del desarrollo basado en modelos se definen en las disciplinas de Requerimientos donde se obtiene el CIM, en la de diseño donde se realizan el PIM y el PIM marcado y en la de implementación donde se genera el PSM y/o el código a partir de los modelos.

Las actividades y entregables definidos para el proceso MDA son independientes de la o las herramientas que se utilicen para ponerlo en práctica, y está basado en la definición del enfoque provista por el estándar de la OMG. Esto permite que pueda ser aplicado utilizando distintas herramientas que cumplan los requerimientos establecidos, lo que abre las puertas para la evaluación de herramientas disponibles. La curva de aprendizaje del enfoque y de la herramienta puede resultar alta al principio del proyecto, pero una vez que se maduran los conceptos y se comprende el funcionamiento de la herramienta, el código generado permite un rápido avance en el desarrollo. Se realizarán evaluaciones de la calidad del código generado, por ejemplo la aplicación de patrones de diseño.

Para el próximo año se piensa conjuntar la Extensión MDA con la Extensión SOA realizada para desarrollos con enfoque Service Oriented Architecture (SOA) en el año 2005 como parte de la tesis de maestría de un docente del Gris según [29], [30] y [31].

6. Referencias

[1] Grupo de Ingeniería. de Software (Gris), Instituto de computación, Facultad de Ingeniería, Universidad de la República http://www.fing.edu.uy/inco/grupos/gris/

[2]Proyecto Ingeniería de Software,Instituto de Computación Facultad de Ingeniería, Universidad de la República http://www.fing.edu.uy/inco/cursos/ingsoft/pis/index.htm

[3] IBM Rational Unified Process. http://www-130.ibm.com/developerworks/rational/products/rup/

[4] Beck K., Extreme Programming Explained: Embrace Change, Addison-Wesley Professional, 1999

[5] Pérez, B. & Delgado, A., Modelo de Desarrollo de Software OO – Experimentación en un curso de Ingeniería de Software, JIISIC 2006, Puebla, México.

[6] Extensión MDA http://www.fing.edu.uy/~pgmdapis

[7] Object Management Group (OMG), Model Driven Architecture (MDA), http://www.omg.org/mda/

[8] Kleppe A., Warmer J., Bast W. MDA Explained, The Model Driven Architecture: practice and promise, Addison-Wesley Pearson Education, 2003

[9] Pérez, B. & Delgado, A., Modelado del proceso de software - Proyecto de Taller V, Instituto de computación, Facultad de Ingeniería, Universidad de la República, 2000

[10]Genexushttp://www2.gxtechnical.com/portal/hgxpp001.aspx?15,8,8,O,E,0,,688

[11] Eclipse, http://www.eclipse.org

[12] King's College London, An evaluation of Compuware OptimalJ Professional Edition as an MDA Tool, 2003

[13]RationalSoftwareArchitect,http://www-128.ibm.com/developerworks/rational/products/rsa/

[14] AndroMDA, http://www.andromda.org

[15] EclipseUML Studio Edition, http://www.omondo.com

[16] Poseidón, http://www.gentleware.com

[17] Magic Draw, http://www.magicdraw.com

[18] Proyecto LIMS, IPMONT-FING http://www.fing.edu.uy/inco/pm/Convenios/LIMS2005

[19] J2EE, http://java.sun.com/j2ee/

[20] Linux http://www.linux.org/

[21] JBoss AS, http://www.jboss.com/products/jbossas

[22] JBoss Portal, http://www.jboss.com/products/

[23] JBPM, http://www.jboss.com/products/jbpm

[24] PostgreSQL, http://www.postgresql.org

[25] Proyecto EBI-PIMS http://www.ebi.ac.uk/ http://www.pims-lims.org/svn/pims/frontpage/index.html

[26] JSF http://java.sun.com/javaee/javaserverfaces/

[27] Spring http://www.springframework.org/

[28] Hibernate http://www.hibernate.org/

[29] Extensión SOA http://www.fing.edu.uy/~adelgado/ExtensionSOA/

[30] Delgado, A., Gonzales L. Piedrabuena F., Enfoque de desarrollo para aplicaciones Service Oriented Architecture (SOA), JIISIC 2006, Puebla, México.

[31] Delgado, A., Metodología de desarrollo para aplicaciones Service Oriented Architecture (SOA), CLEI 2006, Santiago de Chile, Chile.

Agradecimientos

El presente trabajo ha sido desarrollado en el marco del proyecto COMPETISOFT (Mejora de Procesos para Fomentar la Competitividad de la Pequeña y Mediana Industria de Software de Iberoamérica) del programa CYTED (Ciencia y Tecnología para el Desarrollo).

Organización de conocimientos en procesos de ingeniería de software por medio de modelado de procesos: una adaptación de SPEM

Oscar M. Rodríguez-Elias[1,2], Ana I. Martínez García[2], Aurora Vizcaíno[3], Jesús Favela[2], Mario Piattini[3]
[1]*UABC-Facultad de Ciencias, Ensenada, B.C., México*
omrodriguez@ieee.org
[2]*CICESE-Departamento de Ciencias de la Computación, Ensenada, B.C., México*
{martinea | favela}@cicese.mx
[3]*Universidad de Castilla-La Mancha, Escuela de Informática, Ciudad Real, España*
{aurora.vizcaino | mario.piattini}@uclm.es

Resumen

Este trabajo presenta una adaptación del Software Process Engineering Metamodel (SPEM), enfocada en apoyar en el estudio de procesos de ingeniería de software desde una perspectiva del conocimiento involucrado en los mismos. El objetivo es que los modelos de procesos desarrollados, puedan ser de utilidad para el estudio de necesidades de conocimiento por parte de los encargados del proceso. Lo que a su vez puede servir para la definición de estrategias, métodos, sistemas, etc., enfocados en dar solución a las necesidades de conocimiento por medio de la aplicación de técnicas de administración del conocimiento, que estén conectadas con las necesidades reales de quienes desempeñan las actividades dentro de los procesos. La propuesta es ejemplificada por medio de un caso de estudio donde se catalogan algunos de los temas de conocimiento que son relevantes dentro de una parte de un proceso de mantenimiento de software.

Palabras clave: *Modelado de procesos, ingeniería de software, administración de conocimiento, SPEM.*

1. Introducción

La ingeniería de software (IS) es conocida por ser una actividad que involucra distintos tipos de conocimiento, de ahí la necesidad creciente en distintas organizaciones de desarrollo de software para definir métodos y técnicas que les ayuden a manejar el conocimiento que poseen de una mejor manera [1].

La introducción de sistemas de apoyo a la administración del conocimiento (AC) en grupos de IS ha aportado diversos beneficios [2]. Sin embargo, los sistemas que comúnmente son implantados en estas organizaciones poseen características que evitan su uso [3]. Particularmente, este tipo de problemas se deriva de que los sistemas de AC comunes están desconectados de los procesos de trabajo de las organizaciones, y por lo tanto, no están enfocados en resolver sus necesidades de conocimiento reales [4].

Antes de desarrollar o implantar sistemas de AC en un grupo específico, es necesario hacer un estudio de las necesidades de conocimiento del mismo [5], haciendo un análisis de los procesos de trabajo desde un enfoque centrado en cómo fluye y se maneja el conocimiento a través de los procesos realizados por el grupo en cuestión [6]. Un paso importante en este análisis, es el modelado de los procesos [7].

Este trabajo presenta una adaptación hecha al Software Process Engineering Metamodel (SPEM) [8], con el fin de usarlo como apoyo en el estudio de necesidades de conocimiento en grupos de IS. El resto de este artículo se estructura de la siguiente manera: la sección 2 presenta una introducción al trabajo previo en el uso del modelado de procesos para estudiar flujos de conocimiento, y en particular, a su aplicación en procesos de IS. A continuación, en la sección 3 se describen las adaptaciones realizadas a SPEM. Posteriormente, en la sección 4 se da un ejemplo de la aplicación de las extensiones por medio de un caso de estudio. La sección 5 presenta las principales lecciones aprendidas en el caso de estudio. Finalmente, en la sección 6 se resumen las conclusiones de este trabajo.

2. Modelado de procesos con enfoque en el flujo del conocimiento

Para que los sistemas de AC sean realmente de utilidad, es importante que estos ayuden a que las personas encargadas de los procesos de las organizaciones puedan contar con el conocimiento que requieren para llevar a cabo sus actividades [5, 9]. De aquí que, el flujo del conocimiento a través de los procesos de trabajo se vuelve un aspecto central de los sistemas de AC [10]. En este sentido, un primer paso para brindar apoyo por medio de la AC en una organización, es entender cómo fluye el conocimiento a través de sus procesos de trabajo [11]; para posteriormente identificar las necesidades de conocimiento reales y utilizarlas como base para proponer alternativas de solución.

De lo anterior se puede observar cómo el uso de técnicas de análisis de procesos puede ser de gran beneficio al momento de definir estrategias de AC que estén integradas al trabajo real de las organizaciones, y que consideren las necesidades de conocimiento de las mismas [6]. Un medio útil para la identificación de estas necesidades es el modelado de procesos con un enfoque al análisis del flujo del conocimiento [12-14].

El modelado de procesos puede apoyar en la identificación del conocimiento que entra y sale de las actividades de un proceso, las fuentes donde es obtenido o almacenado, y los flujos del conocimiento a través de actividades, personas, u otro tipo de fuentes [15, 16]. Además, a través del modelado es posible detectar los problemas que puedan estar afectando al flujo del conocimiento, facilitando, a su vez, la definición de estrategias para mejorarlo [12, 14].

Existen una gran variedad de técnicas de modelado de procesos, cada una de las cuales se enfoca en aspectos específicos del proceso, e incluso en tipos de procesos particulares [7, 17]. Es importante que la técnica empleada para el estudio de un proceso desde la perspectiva del flujo del conocimiento, cuente con primitivas que faciliten la representación del conocimiento usado, creado, etc. durante las actividades [13]. Sin embargo, dado que la mayoría de las técnicas han sido definidas con el propósito de la reingeniería de procesos de negocios, por lo general no cuentan con los elementos que permitan representar el conocimiento involucrado en el proceso desde la perspectiva del flujo del conocimiento [13, 18].

Debido a lo anterior se han propuesto algunas técnicas para modelado de procesos, con un enfoque al conocimiento involucrado en el mismo, que han sido empleadas en diversos campos [18-21]. Si bien algunas de ellas han sido usadas en organizaciones de desarrollo de software, estas no han sido definidas específicamente para procesos de IS.

2.1 Modelado de procesos de ingeniería de software

Curtis et al. [7] mencionan que mucha de la investigación en modelado de procesos ha sido realizada en organizaciones de desarrollo de software debido a que la comunidad de IS está acostumbrada al modelado formal. Incluso varios de los lenguajes que han sido usados para modelado de procesos se han derivado de lenguajes usados para análisis, diseño e incluso programación de sistemas de software [7, 17].

Por otra parte, existen distintos trabajos enfocados en identificar el conocimiento requerido por los encargados del desarrollo y mantenimiento de software [22-25]. Sin embargo, pocos trabajos han aplicado técnicas de ingeniería de procesos para identificar requerimientos de conocimiento en grupos específicos. No obstante que para aplicar la AC en un grupo en particular, es importante identificar las necesidades y el contexto de dicho grupo [9].

La técnica de gráfica rica [26], es una de las pocas de uso extendido que ha sido usada para analizar flujos de conocimiento en procesos de IS [12, 14]. Esta técnica es muy útil para obtener una visión general del proceso, dado que por su generalidad, es posible modificarla según las necesidades. Sin embargo, la falta de formalidad en los modelos tiene el inconveniente de que se dificulta la identificación de algunos detalles de bajo nivel que podrían ser importantes, es por eso que se recomienda usarla en etapas tempranas del modelado, y complementarla posteriormente con técnicas más formales [26, 27].

Otros trabajos como [20, 21, 28] han empleado sus propios enfoques de modelado de procesos, e incluso proponen herramientas para apoyar en el modelado de los mismos. Sin embargo, ninguna de estas ha sido desarrollada para procesos de IS, además, su utilización requiere que se tengan disponibles las herramientas de modelado que proponen. Otros enfoques usados son los derivados de la ingeniería del conocimiento [29]; el problema de estas técnicas es que no necesariamente son de ayuda para identificar posibilidades de mejoras en los procesos [13].

Si bien el uso de técnicas genéricas de modelado de procesos puede ser de ayuda para estudiar flujos de conocimiento, es recomendable proveer elementos de modelado que permitan representar aspectos particulares del tipo de proceso estudiado, debido a que la representación explícita de estos elementos facilita su análisis [17]. En este sentido, para estudiar flujos de conocimiento en grupos de desarrollo y mantenimiento

de software, es recomendable usar una técnica de modelado de procesos que esté enfocada a procesos de IS, y que a su vez cuente con primitivas que permitan la representación explícita del conocimiento involucrado en el proceso.

2.2 SPEM

SPEM es un metamodelo diseñado para describir procesos y sus componentes, siguiendo un enfoque de modelado orientado a objetos con base en UML [8]. SPEM ha sido desarrollado como un perfil de UML; es decir que extiende los mecanismos de UML de una forma estandarizada, con el propósito de modelar procesos de desarrollo de software.

Existen varias ventajas con respecto al uso de SPEM como lenguaje de modelado de procesos de IS. Primero, SPEM ha sido desarrollado por el Object Management Group (OMG), lo que le da el potencial de convertirse en un lenguaje estándar para el modelado de procesos de IS.

En segundo lugar, UML es el lenguaje de modelado más extendido en las organizaciones de desarrollo de software, lo que puede facilitar la asimilación de SPEM como lenguaje de modelado de procesos en este tipo de organizaciones. Esto es importante si consideramos que un estudio de procesos requiere de diversas iteraciones, lo que implica que se requiere retroalimentación de los participantes del proceso para ir mejorando los modelos en cada iteración [30]. Por lo tanto, usar un lenguaje con el que puedan estar familiarizados podría facilitar esta labor.

Finalmente, para modelar procesos con SPEM es posible usar cualquier herramienta que permita el uso de perfiles de UML [31]. Dado el extendido uso de UML, existen muchas herramientas disponibles con esta característica. Incluso, interfaces ampliamente usadas como plataformas de desarrollo de software, comienzan a proveer facilidades para modelar procesos con base en SPEM, tal es el caso del proyecto Eclipse Process Framework, (http://www.eclipse.org/epf/).

Sin embargo, SPEM no provee elementos para representar el conocimiento involucrado en los procesos. Con el fin de usarlo para estudiar flujos de conocimiento en procesos de IS, fue necesario adaptarlo añadiendo estos elementos como parte del lenguaje. A continuación se describen las adaptaciones que, siguiendo este enfoque, se hicieron a SPEM.

3. Adaptación de SPEM para estudiar flujos de conocimiento

La adaptación que se hizo de SPEM está basada en tres aspectos principales:

(i) Ilustrar el conocimiento, y sus fuentes, involucrado (usado, creado, y/o modificado) en las actividades del proceso.
(ii) Ilustrar flujos de conocimiento entre actividades, y cómo las fuentes que lo contienen son usadas o modificadas a través de éstas.
(iii) Ilustrar las transferencias de conocimiento entre roles u otros tipos de fuentes.

Siguiendo estas tres premisas, se definieron una serie de conceptos que se agregaron como parte del metamodelo de SPEM, y se propuso un conjunto de elementos para complementar el lenguaje de modelado. Estos elementos se describen a continuación.

3.1 El conocimiento como producto del trabajo

SPEM basa su notación en tres elementos básicos: productos del trabajo (*work products*), definiciones de trabajo (*work definitons*), y roles (*roles*). Las definiciones de trabajo son operaciones que describen el trabajo realizado por los roles del proceso. Estas son usadas para estructurar el proceso, por ejemplo, describiendo su ciclo de vida, sus fases, iteraciones, o actividades. Los productos del trabajo es todo aquello usado o generado durante el proceso.

Con base en lo anterior, si consideramos al conocimiento como un recurso que es usado, generado o modificado durante las actividades del proceso, entonces podemos definirlo como un tipo de producto del trabajo. La figura 1 muestra los elementos principales que se agregaron a SPEM para permitir la representación del conocimiento en los modelos.

En particular, se definió un tipo de producto de trabajo para referirse al conocimiento involucrado en las actividades. Este conocimiento puede ser conceptos específicos, como temas o fuentes de conocimiento; o paquetes de conocimiento, que son grupos de temas de conocimiento relacionados, por ejemplo, pueden usarse para agrupar el conocimiento de un rol determinado, de un documento, etc. Otro uso que se puede dar a estos paquetes de conocimiento, es para agrupar temas específicos en temas más generales, en áreas, o en categorías, con el fin de usarlos para estructurar y clasificar el conocimiento, como se ejemplifica más adelante en este trabajo.

3.2 Relaciones

A la par de los conceptos para definir el conocimiento involucrado en las actividades, se definió un conjunto de relaciones para definir transferencias de conocimiento, así como el conocimiento que puede estar contendido en fuentes determinadas.

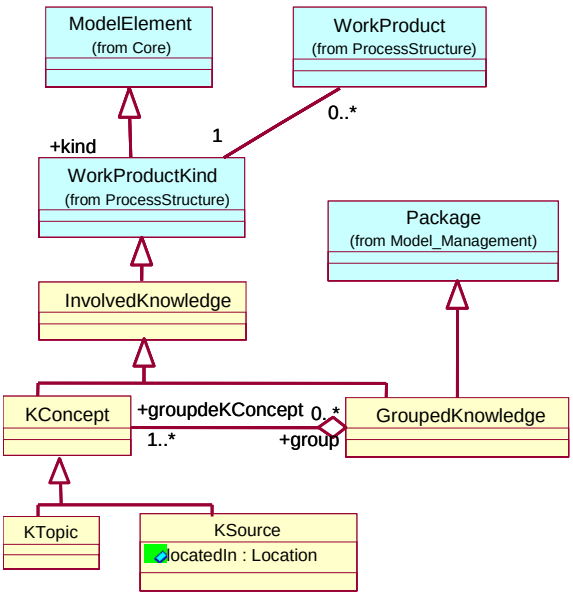


Fig. 1. Integración de los conceptos de conocimiento al metamodelo SPEM.

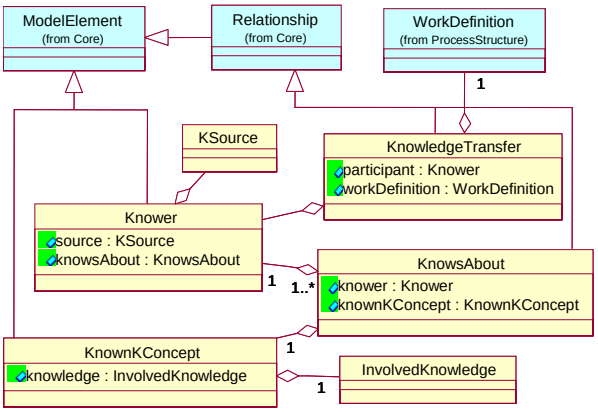


Fig. 2. Relaciones KnowledgeTransfer y KnowsAbout.

La figura 2 ilustra las relaciones que han sido definidas, y cómo estas se relacionan al metamodelo de SPEM. La relación *KnowledgeTransfer* es usada para representar transferencias de conocimiento entre fuentes, por ejemplo entre dos o más roles, entre un rol y un documento, etc. Esta relación tiene dos propiedades principales, las fuentes involucradas en la transferencia, y la definición de trabajo donde ésta se da. Los conocimientos transferidos u obtenidos por una fuente pueden ser temas específicos, o grupos de estos temas, y se especifican con una relación tipo *KnowsAbout*. La relación KnowsAbout define el conocimiento contenido en una determinada fuente.

3.3 Notación

Para integrar los elementos antes mencionados, en los diagramas generados con SPEM, se definieron una serie de iconos mostrados en la Figura 3. Estos iconos son usados en conjunto con los ya definidos por SPEM. Los temas de conocimiento o habilidades específicas son representados con elementos tipo KTopic. El elemento GroupedKnowledge es usado para representar grupos de temas relacionados, a su vez, es también útil para clasificar estos temas en paquetes. Finalmente, las transferencias de conocimiento son representadas por medio del icono KnowledgeTransfer.

Fig. 3. Notación usada para representar los conceptos de conocimiento en diagramas de SPEM.

Con excepción del icono KnowledgeTransfer, el resto es usado en los diagramas predefinidos en SPEM. Para la representación de transferencias de conocimiento hemos definido diagramas adicionales. Para representar el proceso de manera general, así como las principales fuentes, o paquetes de conocimiento que intervienen, empleamos los casos de uso. Los diagramas de actividad nos ayudan a identificar detalles del conocimiento involucrado en actividades específicas, así como el conocimiento que cada rol puede requerir al desempeñar dichas actividades. Los diagramas de clases permiten representar el conocimiento que puede ser obtenido de cada fuente, así como dependencias y relaciones entre tipos de conocimiento y/o fuentes. Los diagramas de paquetes son usados para organizar y clasificar el conocimiento o habilidades en paquetes de conocimiento relacionados, por ejemplo, el conocimiento contenido en una fuente determinada. Finalmente, el análisis de transferencias de conocimiento es realizado por medio de diagramas de transferencia, que son un tipo de diagrama que se ha definido con base en la relación KnowledgeTransfer. Este tipo de diagramas permiten ilustrar las fuentes participando en la transferencia, la actividad o flujo de trabajo donde se da la transferencia, el conocimiento que es transferido, así como el conocimiento que cada fuente aporta, y el que obtienen al participar en dicha transferencia. Si se requiere especificar la transferencia con mayor detalle, por ejemplo, protocolos de discusión entre roles, o secuencia de llenado de contenido en un documento, etc., se hace uso de los diagramas de secuencia, o de diagramas de estados.

A continuación ilustramos el uso de los elementos de modelado por medio de un caso de estudio. En particular, el caso de estudio ejemplifica cómo el uso de los elementos de modelado puede ayudar a organizar y clasificar los temas de conocimiento involucrados en las actividades llevadas a cabo por el grupo estudiado.

4. Caso de estudio

Con el fin de aplicar estrategias de AC en una organización, uno de los primeros pasos es identificar las necesidades reales de conocimiento de las personas involucradas en los procesos de la misma [5, 6]. Esto puede llevarse a cabo por medio de una "auditoria de conocimiento" que nos ayude a identificar qué conocimiento es requerido por los participantes del proceso, qué fuentes de conocimiento existen, y qué conocimiento puede ser obtenido de dichas fuentes [32]. Para facilitar el análisis y manejo de esta información, una técnica muy útil es la definición de taxonomías de conocimientos y fuentes, las cuales nos permitirán clasificar estos elementos de una manera que se facilite la estructuración y generación de la base de conocimientos de la organización [32].

En el caso de estudio presentado en esta sección, se ejemplifica cómo el uso de los elementos de modelado descritos, permiten identificar el conocimiento y sus fuentes. A su vez se ilustra cómo desde el modelado de los procesos es posible comenzar a definir una clasificación de dichos elementos, con el fin de que ésta sirva para definir las taxonomías de conocimientos y fuentes que posteriormente puedan ser usadas para estructurar y generar la base de conocimientos del proceso.

Para este ejemplo se ha tomado información obtenida en un grupo dedicado al mantenimiento del software de un centro de investigación en México (ver [14]).

4.1 Diagramas de actividad y paquetes de conocimientos

La figura 4 muestra un ejemplo del uso de los diagramas de actividad para representar el conocimiento que es usado y generado en las actividades. El diagrama describe parte de las actividades que realiza un ingeniero de software durante el proceso de mantenimiento estudiado. Como se puede observar, el conocimiento que cada fuente

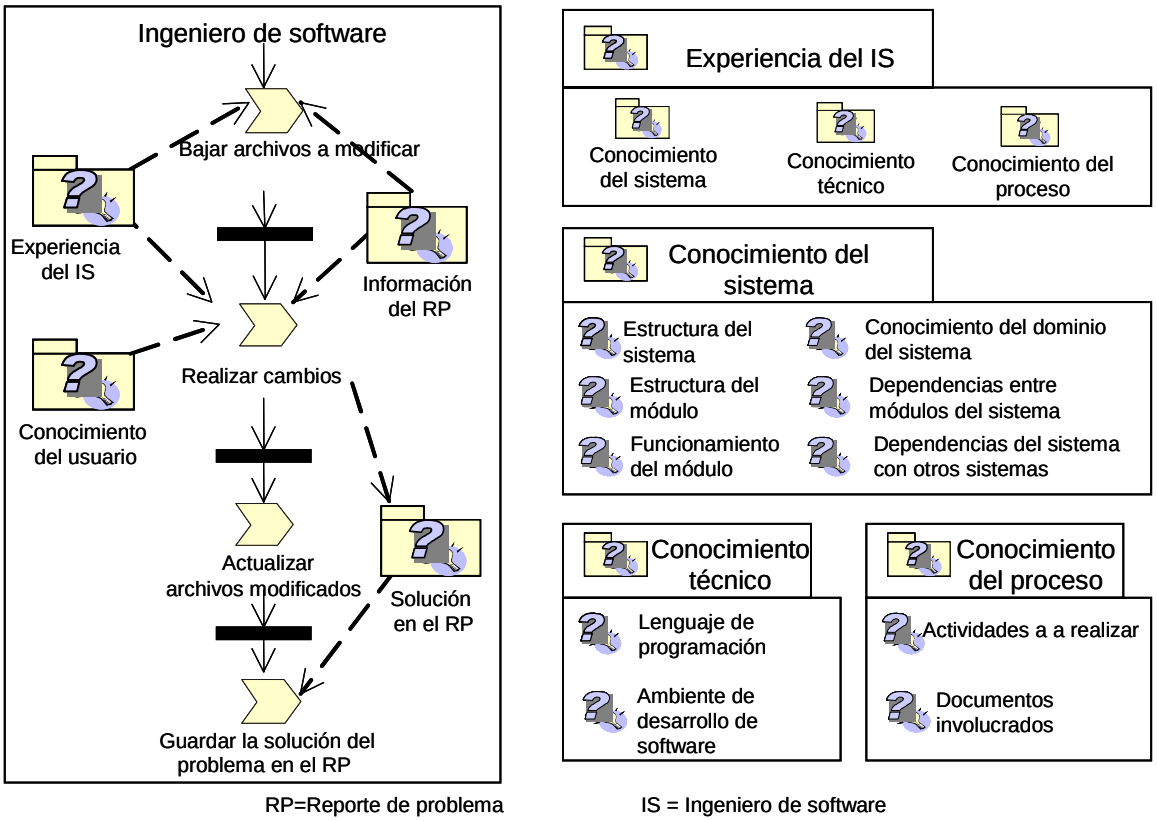


Fig. 4. Ejemplo de un diagrama de actividad que muestra el conocimiento involucrado en las actividades.

aporta u obtiene es agrupado en un paquete que representa el conocimiento de esa fuente. Posteriormente, estos paquetes de conocimiento son extendidos para definir el conocimiento específico que agrupan. Este conocimiento puede ser temas concretos, o paquetes que agrupan temas relacionados. De esta manera, al ir modelando el proceso es posible comenzar la clasificación del conocimiento involucrado, a la vez que se comienza la identificación de las fuentes que pueden contener el conocimiento, ya sea porque son las fuentes de donde es obtenido, o donde es almacenado.

A partir de modelos genéricos como el mostrado en la figura 4, se inició la identificación de temas y fuentes de conocimiento específicos. Como ejemplo veamos el caso del conocimiento sobre un sistema específico, digamos sistema A, que es un sistema para control de los alumnos inscritos en los posgrados que se imparten en el centro de investigación.

El conocimiento del dominio del sistema A, comprende los procesos escolares, tales como procesos de inscripciones, control de becas a estudiantes, etc. Este tipo de información está contenida en diversas fuentes, entre las que destacan los reglamentos y normas escolares, así como el personal de la dirección a cargo del control de estudiantes. Al identificar los distintos procesos a los que da soporte el sistema, se facilitó la identificación de las fuentes específicas que pueden servir para obtener información sobre dichos procesos. De esta manera se comenzó la estructuración y clasificación de dichas fuentes como parte de la base de conocimientos para apoyar el mantenimiento del sistema de control de estudiantes.

Por medio de este análisis, fue posible también identificar la estrecha relación que guardan las fuentes de conocimiento sobre el dominio del sistema A, con la evolución del mismo sistema. Esto se debe a que los principales cambios en el sistema A se derivan de cambios en los reglamentos escolares, y es en base a estos reglamentos, y en la información proporcionada por el personal a cargo de los procesos escolares, que se definen los cambios que deben hacerse al sistema A, y la manera de hacerlos.

Con relación a la estructura del sistema A, se detectó un documento que describe las relaciones entre los distintos módulos del sistema y los archivos fuente que corresponden con dichos módulos. Esta fuente de conocimiento resulta de gran ayuda a la hora de hacer modificaciones al sistema, sobre todo si éstas requieren ser hechas en módulos del sistema que no son totalmente conocidos por la persona encargada de realizar los cambios.

Con respecto al funcionamiento de los distintos módulos del sistema A, se detectaron documentos que describen la manera en que deben operar. Dichos documentos han sido generados para apoyar a los usuarios, pero también resultaron ser una útil fuente de información para los encargados de modificar el sistema, dado que por medio de dichos documentos pueden darse una idea de cómo debe funcionar cada módulo. Las relaciones entre fuentes y el conocimiento que puede ser obtenido de ellas se modeló por medio de diagramas de clases, como se describe a continuación.

4.2 Diagramas de clases: relaciones entre el conocimiento y sus fuentes

La figura 5 presenta un ejemplo del uso de diagramas de clases para describir las relaciones entre fuentes y conocimiento. Este tipo de diagrama es usado para describir el conocimiento que cada fuente contiene, la forma en que están agrupadas las fuentes,

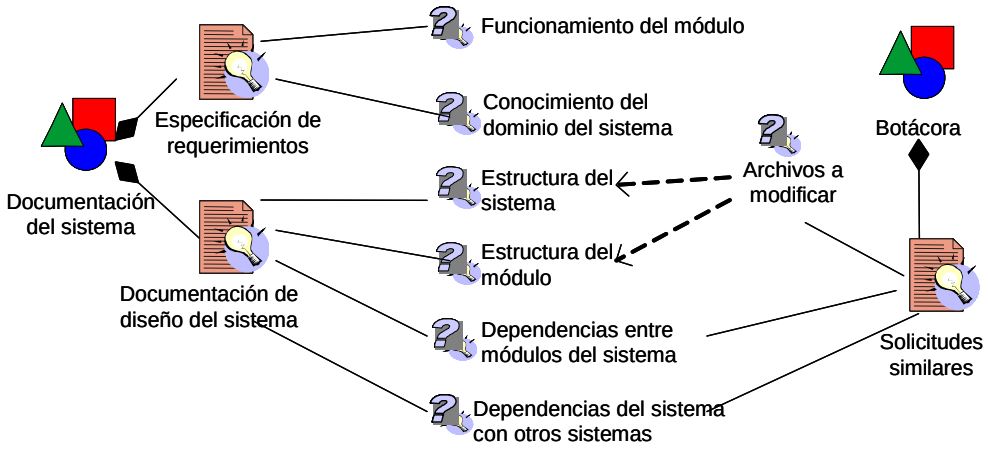


Fig. 5. Ejemplo de un diagrama de clases que ilustra el conocimiento que puede ser obtenido de cada fuente.

así como relaciones y dependencias entre tipos de conocimiento. Por ejemplo, en la figura 5 se muestra que las solicitudes de mantenimiento similares a la que se está atendiendo pueden ser usadas para obtener información sobre los archivos que requerirán ser modificados. A su vez, para esto también es necesario conocer la estructura del sistema y del módulo específico que se requiere modificar, información que puede ser obtenida del documento de diseño del sistema.

Por medio de las extensiones propuestas a SPEM, también es posible crear diagramas que se enfocan en identificar transferencias de conocimientos entre actividades, roles y otro tipo de fuentes. El objetivo del modelado de estos flujos de conocimiento es identificar los problemas que pudieran estar afectando que el conocimiento fluya adecuadamente a través del proceso. Ejemplos de lo anterior son la identificación de conocimiento que es generado en ciertas actividades y no es guardado, aun cuando sea importante para realizar otras actividades. Otro caso es la identificación de fuentes importantes con respecto al conocimiento que contienen, y que no están siendo usadas adecuadamente durante el proceso.

Por ejemplo, en el grupo estudiado identificamos que los requerimientos relacionados con las solicitudes de modificaciones a los sistemas, no están siendo capturados adecuadamente, ya que existe un mecanismo estándar para esto. De manera que se dificulta saber con posterioridad qué fue lo que generó un cambio determinado.

La identificación de este tipo de situaciones es importante debido a que pueden contribuir a proponer alternativas para mejorar los flujos de conocimiento durante el proceso. Si consideramos que los procesos de desarrollo y mantenimiento de software utilizan gran cantidad de conocimiento, entonces lograr esquemas que permitan que el conocimiento fluya de forma más rápida y eficiente, puede contribuir en gran medida a la mejora de dichos procesos.

5. Lecciones aprendidas

Las siguientes son dos de las principales lecciones aprendidas en la realización de este trabajo:

Identificación de problemas relacionados con el conocimiento. La aplicación de SPEM, y en particular el uso de las extensiones definidas para representar explícitamente el conocimiento involucrado en los procesos modelados, permitieron identificar problemas en los flujos de conocimiento que actualmente se dan dentro del proceso. Por ejemplo, se identificó información y conocimiento que no estaba siendo almacenado por el grupo, aun cuando resultaba de gran relevancia para otras actividades, o para otros roles dentro del proceso. Esto fue posible gracias a la representación explícita del conocimiento y fuentes involucradas en los procesos. De no haber contado con la representación explícita del conocimiento en los diagramas de SPEM, parte de estos problemas difícilmente hubieran sido identificados.

Estructuración y clasificación de temas y fuentes de conocimientos. La posibilidad de representar, tanto temas específicos como conjuntos de los mismos, facilitó la definición de una estructura de clasificación de conocimientos y fuentes, lo cual ayudó a estructurar un mapa de conocimientos del proceso estudiado [33].

Los dos puntos mencionados son importantes debido a que antes de realizar la mejora de cualquier proceso, es necesario identificar las debilidades del mismo. Por otra parte, un primer paso en la definición de sistemas de AC, es definir taxonomías que permitan estructurar, clasificar y organizar el conocimiento de la organización y las fuentes que lo contienen [32].

Como resultado de este trabajo, el grupo donde se realizó el estudio actualmente se encuentra tomando medidas para resolver parte de los problemas observados, con el fin de mejorar su proceso de desarrollo y mantenimiento de software.

6. Conclusiones y comentarios finales

Con base en nuestra experiencia en este trabajo, podemos constatar lo que otros autores han afirmado con respecto a la necesidad de proveer lenguajes de modelado de procesos que estén orientados a representar el conocimiento involucrado en los procesos [13, 18]. Consideramos que SPEM es una buena alternativa como lenguaje para analizar los procesos existentes en grupos de IS. Debido a la alta dependencia del conocimiento que existe en este tipo de procesos, un lenguaje de modelado que considere el conocimiento involucrado podría ser de gran ayuda.

Ya que SPEM no cuenta con primitivas para representar explícitamente el conocimiento involucrado en procesos de IS, vimos la necesidad de dotarlo con esta alternativa con el fin de que sirva para analizar y definir procesos de IS considerando el conocimiento involucrado en las actividades que constituyen a los procesos. El trabajo presentado en este artículo es un esfuerzo en este sentido. La experiencia obtenida al aplicarlo a un caso real, nos ha dado evidencia de que puede ser de gran ayuda para apoyar en la definición de estrategias de AC en grupos encargados del desarrollo y mantenimiento de software.

En particular, el uso de las extensiones definidas para representar el conocimiento involucrado en el

proceso, fue de gran ayuda para identificar problemas relacionados con el conocimiento, y de esta manera comenzar a tomar medidas para abordarlos. Así también, el uso de las extensiones permitió definir modelos generales del proceso de mantenimiento del grupo estudiado junto con las principales áreas o temas generales de conocimiento involucrados. Esto facilitó la identificación de temas o áreas de conocimiento específicos relacionados con cada uno de los sistemas que son mantenidos por el grupo. Por ejemplo, los temas de conocimiento relacionados con el dominio de cada uno de dichos sistemas.

Como trabajo futuro, buscaremos integrar las adaptaciones hechas a SPEM en un ambiente de modelado de procesos. De forma tal que los modelos sirvan también para definir la estructura de la base de conocimientos del proceso. Así mismo, seguiremos aplicando el enfoque de modelado para estudiar otros procesos y otros grupos de desarrollo, con el fin de continuar nuestra evaluación de las adaptaciones a SPEM que han sido definidas, y extenderlas o mejorarlas en caso de ser necesario.

7. Agradecimientos

Este trabajo fue apoyado en parte por CONACYT bajo el proyecto C01-40799 y la beca 164739 proporcionada al primer autor, en México; y por los proyectos MECENAS (PBI06-0024) y ENIGMAS (PIB-05-058), de la Junta de Comunidades de Castilla-La Mancha, Consejería de Educación y Ciencia, en España. Este trabajo también forma parte de la red CALIPSO (TIN2005-24055-E), apoyada por el Ministerio de Ciencia y Tecnología en España.

8. Referencias

[1] A. Aurum, R. Jeffery, C. Wohlin, and M. Handzic, "Managing Software Engineering Knowledge." Springer, Berlin, Germany, 2003.

[2] T. Dingsøyr and R. Conradi, "A survey of case studies of the use of knowledge management in software engineering," International Journal of Software Engineering and Knowledge Engineering, vol. 12, num. 4, 2002, pp. 391-414.

[3] K. C. Desouza, "Barriers to Effective Use of Knowledge Management Systems in Software Engineering," Comm. of the ACM, 46(1), 2003, pp. 99-101.

[4] T. A. Stewart, "The Case Against Knowledge Management," Business 2.0, vol. 3, February, 2002, pp. 80.

[5] K. Wiig, People-Focused Knowledge Management: How Effective Decision Making Leads to Corporate Success. Elsevier, Amsterdam, 2004.

[6] R. Maier and U. Remus, "Defining Process-oriented Knowledge Management Strategies," Knowledge and Process Management, vol. 9, num. 2, 2002, pp. 103-118.

[7] B. Curtis, M. I. Kellner, and J. Over, "Process Modeling," Comm. of the ACM, vol. 35, num. 4, 1992, pp. 75-90.

[8] "Software Process Engineering Metamodel Specification (SPEM)," vol. 2004. Object Management Group, 2002.

[9] K. Dalkir, Knowledge Management in Theory and Practice. Elsevier, Amsterdan, 2005.

[10] U. M. Borghoff and R. Pareschi, "Information Technology for Knowledge Management," Journal of Universal Computer Science, vol. 3, num. 8, 1997, pp. 835-842.

[11] M. E. Nissen, "An Extended Model of Knowledge-Flow Dynamics," Communications of the Association for Information Systems, vol. 8, 2002, pp. 251-266.

[12] B. H. Hansen and K. Kautz, "Knowledge Mapping: A Technique for Identifying Knowledge Flows in Software Organizations", Proc. of the European Conference on Software Process Improvement (EuroSPI 2004), Trondheim, Norway, 2004, pp. 126-137.

[13] P. Bera, D. Nevo, and Y. Wand, "Unraveling Knowledge Requirements through Business Process Analysis," Comm. of the AIS, vol. 16, 2005, pp. 814-830.

[14] O. M. Rodríguez, A. I. Martínez, A. Vizcaíno, J. Favela, and M. Piattini, "Identifying Knowledge Management Needs in Software Maintenance Groups: A qualitative approach", Proc. of the Fifth Mexican International Conference on Computer Science (ENC'2004), Colima, México, 2004, pp. 72-79.

[15] M. S. Abdullah, I. Benest, A. Evans, and C. Kimble, "Knowledge Modelling Techniques for Developing Knowledge Management Systems", Proc. of the European Conference on Knowledge Management, Dublin, Ireland, 2002, pp. 15-25.

[16] T. H. Davenport and L. Prusak, Working Knowledge: How Organizations Manage What they Know. Harvard Business School Press, Boston, Massachusetts, USA, 2000.

[17] R. Conradi and L. Jaccheri, "Process Modelling Languages," LNCS 1500, Springer, Berlin, 1999, pp. 27-52.

[18] G. Papavassiliou and G. Mentzas, "Knowledge modelling in weakly-structured business processes," Journal of Knowledge Management, vol. `7, num. 2, 2003, pp. 18-33.

[19] S. Kim, H. Hwang, and E. Suh, "A Process-based Approach to Knowledge Flow Analysis: A Case Study of a manufacturing Firm," Knowledge and Process Management, vol. 10, num. 4, 2003, pp. 260-276.

[20] M. Strohmaier and K. Tochtermann, "B-KIDE: A Framework and a Tool for Buisiness Process-Oriented Knowledge Infrastructure Development," Journal of Knowledge and Process Management, vol. 12, num. 3, 2005, pp. 171-189.

[21] R. Woitsch and D. Karagiannis, "Process-oriented Knowledge Management Systems based on KM-Services: The PROMOTE Approach," International Journal of Intelligent Systems in Accounting, Finance & Management, vol. 11, 2002, pp. 253-267.

[22] T. C. Lethbridge, "What Knowledge Is Important to a Software Professional?" IEEE Computer, vol. 33, num. 5, 2000, pp. 44-50.

[23] P. N. Robillard, "The Role of Knowledge in Software Development," Comm. of the ACM, vol. 42, num. 1, 1999, pp. 87-92.

[24] K. M. Oliveira, N. Anquetil, D. M.G, M. Ramal, and R. Meneses, "Knowledge for Software Maintenance." Proc. of the Fifteenth International Conference on Software Engineering and Knowledge Engineering (SEKE'03), San Francisco, CA, 2003, pp. 61-68.

[25] M. G. B. Dias, N. Anquetil, and K. M. d. Oliveira, "Organizing the Knowledge Used in Software Maintenance," Journal of Universal Computer Science, vol. 9, num. 7, 2003, pp. 641-658.

[26] A. Monk and S. Howard, "The Rich Picture: A Tool for Reasoning About Work Context," Interactions, vol. 5, num. 2, 1998, pp. 21-30.

[27] P. Checkland and J. Scholes, Soft System Methodology in Action. John Wiley and Sons, 1999.

[28] M. E. Nissen and R. E. Levitt, "Agent-Based Modeling of Knowledge Flows: Illustration from the Domain of Information Systems Design", Proc. of the Hawaii International Conference on System Science (HICSS 2004), Big Island, Hi, USA, 2004, pp.

[29] H. Zhuge, "Knowledge flow management for distributed team software development," Knowledge-Based Systems, vol. 15, num. 8, 2002, pp. 465-471.

[30] D. G. Wastell, P. White, and P. Kawalek, "A Methodology for Business Process Redesign: Experiences and Issues," Journal of Strategic Information Systems, vol. 3, num. 1, 1994, pp. 23-40.

[31] J. Bézivin and E. Breton, "Applying the Basic Principles of Model Engineering to the Field of Process Engineering," UPGRADE, vol. V, num. 5, 2004, pp. 27-33.

[32] M. Rao, "Knowledge Management Tools and Techniques: Practitioners and Experts Evaluate KM Solutions." Elsevier, Amsterdam, 2005, pp. 438.

[33] O. M. Rodríguez-Elias, A. I. Martínez-García, A. Vizcaíno, J. Favela, and M. Piattini, "Constructing a Knowledge Map for a Software Maintenance Organization", Proc. of the Poster Session of the 21st IEEE International Conference on Software Maintenance (ICSM 2005), Budapest, Hungary, 2005, pp. 51-54.

[34] J. P. Soto, O. M. Rodríguez, A. Vizcaíno, M. Piattini, and A. I. Martínez-García, "Localización de fuentes del conocimiento en el proceso del mantenimiento del software," Memorias del Campus Multidisciplinar en Percepción e Inteligencia (CMPI-2006), vol. I. Universidad de Castilla-La Mancha, Albacete, España, 2006, pp. 118-123.

Un modelo de Componentes para el Diseño y Ejecución de Procesos de Colaboración basado en ThinkLets

Víctor Alberto Hermida[1], Carlos Hernán Tobar[1], Julio Ariel Hurtado[1,2], César A. Collazos[1]
[1]*Grupo IDIS, Universidad del Cauca, Colombia*
[2]*Departamento de Ciencias de la Computación, Universidad de Chile, Chile*
{vhermida, carlost, ahurtado, ccollazo}@unicauca.edu.co

Abstract

En muchas tareas las personas unen esfuerzos para alcanzar una meta común. Por medio de la colaboración ellos pueden lograr más de lo que harían como individuos aislados. Los procesos de colaboración se deben diseñar, estructurar y gestionar explícitamente para maximizar la efectividad del grupo. La Ingeniería de la Colaboración es un área de investigación que fomenta el modelado, diseño y despliegue de procesos de colaboración repetibles para que sean ejecutados por los miembros de un grupo. Uno de los actuales focos de investigación en esta área es el concepto de thinkLet, un elemento clave para diseñar procesos de colaboración repetibles y predecibles. Los procesos colaborativos pueden soportarse en sistemas software que faciliten su diseño y ejecución. En este sentido, este artículo presenta una aproximación para crear un framework de componentes que permita el desarrollo de aplicaciones de diseño y despliegue de procesos de colaboración, y muestra su uso en un proceso colaborativo en el contexto de desarrollo de software.

1. Introducción

Las personas frecuentemente se enfrentan a tareas que necesitan ser realizadas por un grupo de trabajo en una manera colaborativa, debido a la naturaleza de las tareas que imponen retos y complejidad que impide a un único individuo realizarlas por sí sólo. La colaboración es la unión de esfuerzos intelectuales lograda por cada miembro de el grupo de trabajo hacia una meta [1]. Las organizaciones, por lo tanto, existen para dar valor a la comunidad de actores que de otra forma no puede ser logrado por ellos mismos como individuos [2].

Cuando las personas trabajan juntas hacia una meta, se mueven a través de procesos en los cuales deben combinar su experticia, su percepción y sus recursos para la realización de las tareas asignadas [3]. Algunas tareas organizacionales son ad hoc, esfuerzos que se realizan una sola vez para un propósito particular, que raramente o nunca se repiten. Sin embargo, otras tareas organizacionales son recurrentes, es decir se repiten frecuentemente. La Ingeniería de la Colaboración es una aproximación para el diseño de procesos y tecnologías colaborativos reusables, que pretende generar tareas colaborativas de misión crítica recurrentes, de forma predecible y exitosa entre los miembros de un grupo de trabajo [3].

Uno de los actuales focos de investigación en la ingeniería de la colaboración es la identificación y documentación de bloques de construcción reusables para el diseño de procesos de colaboración grupales. Hasta el momento, los investigadores han codificado una colección de estos bloques de construcción, llamados ThinkLets [1]. Un thinkLet es una actividad de razonamiento empaquetada y que posee un nombre, que crea un patrón de colaboración predecible y repetible entre las personas que trabajan hacia una meta común [4]. Los thinkLets pueden ser usados como piezas conceptuales en el diseño de procesos de colaboración [5]. Un thinkLet pretende ser la unidad más pequeña de capital intelectual requerido para reproducir un patrón de colaboración entre las personas que trabajan hacia una meta [1].

Este artículo presenta un modelo de componentes thinkLets que permite especificar una infraestructura tecnológica que soporte el uso de thinkLets para la descripción de patrones de colaboración, y que puedan utilizarse como piezas software en el diseño y ejecución de procesos de colaboración. La Sección 2 presenta un modelo basado en thinkLets que sirve como punto de partida para especificar el modelo de componentes. En la Sección 3 se mapea explícitamente el modelo conceptual a un modelo de componentes buscando establecer un framework tecnológico que ayude en el diseño de procesos de colaboración. La Sección 4 presenta una discusión acerca del uso de componentes thinkLet en la captura de requisitos. Finalmente se presentan los resultados de esta investigación y se plantea el trabajo futuro.

2. Trabajos Relacionados

Existen diversas propuestas de modelos y arquitecturas de componentes para soportar el trabajo colaborativo. El enfoque de componentes ha tenido gran aceptación gracias a los beneficios de la reutilización y la definición de formas estándares para la creación de componentes y su composición para formar aplicaciones.

En [6] se propone un framework en el cual los componentes de datos compartidos de acuerdo a una interfaz bien definida pueden ser dinámicamente ensamblados en forma flexible. Bajo este framework la mayoría de componentes en la plataforma estándar de Java (JDK) pueden transformarse en componentes compartidos para el desarrollo de herramientas de colaboración.

En [7] se describe una arquitectura genérica para la implementación de aplicaciones de colaboración. Se presentan varias dimensiones a lo largo de las cuales las arquitecturas se diferencian, y se realiza una clasificación de arquitecturas soportadas por diferentes herramientas de colaboración existentes.

En [8] se presenta una arquitectura basada en componentes que permite modificaciones incrementales al sistema. Esta aproximación se construye sobre la noción de beans y tiene como meta incrementar la flexibilidad de la infraestructura.

En [9] Marsic presenta el framewok Manifold para soportar dispositivos heterogéneos, usa una arquitectura multinivel separando la parte de presentación, lógica de dominio y funcionalidad de colaboración. Manifold utiliza los Java Beans para soportar los dispositivos sobre el nivel de aplicación e interacción. Esto provee una arquitectura extensible y flexible de acuerdo a la infraestructura de comunicación.

En [10] Guicking presenta un framework diseñado para aplicaciones groupware síncronas en ambientes heterogéneos -Agilo, este framework está compuesto de tres niveles: nivel de red, nivel de núcleo y nivel de aplicaciones. Agilo es más flexible y adaptable comparado con otros frameworks cuya funcionalidad está totalmente contenida en el nivel de núcleo, esto debido a que el nivel de aplicaciones se basa en componentes software el cual ofrece las siguientes ventajas: 1) los componentes pueden ser fácilmente reusados; 2) los componentes pueden ser configurados y adaptados independientemente lo cual simplifica las pruebas e incrementa la flexibilidad; 3) la interfaz para programación de aplicaciones del núcleo es más compacta y fácil de usar, y 4) el despliegue de las aplicaciones pueden ser instanciado de acuerdo a las capacidades específicas de los dispositivos.

Estos trabajos coinciden en el uso de componentes software para el desarrollo de sistemas colaborativos que posean atributos de calidad como desempeño, disponibilidad, seguridad y modificabilidad. Los componentes son piezas software reutilizables que se especifican a partir de modelos de componentes tales como COM de Microsoft, CCM de CORBA y EJB de Sun Microsystems. Sin embargo los modelos de componentes y aproximaciones arquitecturales para soportar aplicaciones colaborativas no especifican explícitamente tipos de componentes de colaboración y su interacción.

El modelo de componentes propuesto en este artículo encapsula un concepto clave en la Ingeniería de la Colaboración –el ThinkLet, para el desarrollo de aplicaciones de diseño de procesos de colaboración. Los componentes thinkLets podrán ser desplegados sobre contenedores EJB. El modelo de componentes thinkLet favorece el diseño y ejecución de procesos colaborativos recurrentes, mediante los patrones de colaboración que el thinkLet establece.

3. Un modelo de ThinkLets

Esta sección considera el modelo conceptual basado en thinkLets propuesto en [11]. Este modelo representado en la figura 1 se basa en el concepto de colaboración para el soporte de procesos colaborativos, tiene cinco elementos clave:

- ThinkLet
- ThinkLetTransition
- ComposedThinkLet
- CollaborationPattern
- ThinkLetVariability.

Un proceso colaborativo se enfoca en lo que un grupo debe hacer, es decir los pasos que sus miembros deben seguir para solucionar un problema o alcanzar una meta, sin especificar detalladamente cómo se deben realizar cada uno de los pasos. Los patrones de colaboración ofrecen una respuesta conceptual a la pregunta del cómo, un patrón es una manera de razonar acerca de cómo las personas deben moverse a través de fases para lograr sus objetivos. Cada paso en un proceso puede realizarse por medio de actividades que crean un patrón efectivo de colaboración entre los miembros del grupo de trabajo.

No obstante, este entendimiento aún no contiene el nivel de detalle suficiente para que el equipo de trabajo desarrolle un proceso recurrente. Para solucionar este inconveniente es necesario contar con bloques de construcción básicos que puedan ser enlazados para formar el proceso de colaboración deseado. Los thinkLets proveen esta capacidad, encapsulando especificaciones detalladas para la realización de

actividades repetibles y predecibles. Un thinkLet es la unidad más pequeña de capital intelectual requerido para crear un patrón de razonamiento repetible y predecible entre las personas que trabajan hacia una meta común. Hasta la fecha se han identificado cinco patrones de razonamiento [1]: Divergencia, moverse de tener pocos conceptos a tener más conceptos; Convergencia, pasar de tener muchos conceptos a enfocarse en un subconjunto que amerite mayor atención; Organización, moverse de un menor entendimiento a uno mayor acerca de las relaciones entre conceptos; Evaluación: pasar de un menor entendimiento del valor de los conceptos para alcanzar una meta a un entendimiento mayor, y Construcción de Consenso: alcanzar un mayor acuerdo entre los miembros del grupo de trabajo.

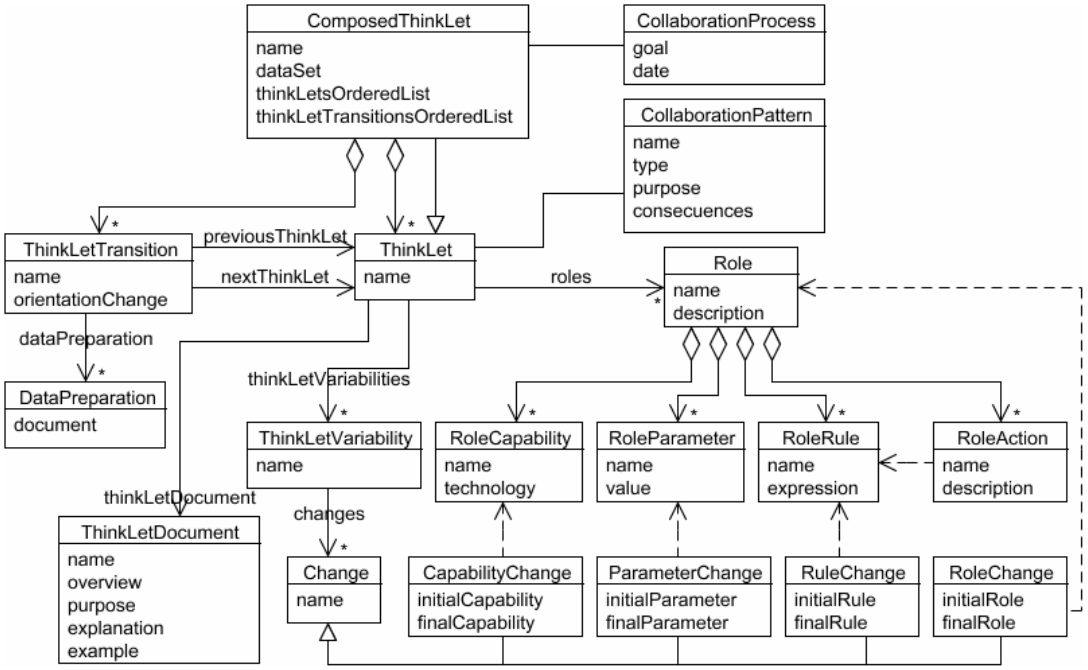


Figura 1. Modelo conceptual para el diseño de procesos de colaboración

Un thinkLetVariability es un conjunto de variabilidades empaquetadas y que poseen un nombre, estas variabilidades pueden aplicarse a uno o más thinkLets para producir un cambio predecible en el patrón que genera (divergencia, convergencia, organización, evaluación y construcción de consenso).

Los thinkLetTransitions son los enlaces que conectan un thinkLet al siguiente. Una transición se define como todo aquello que es necesario para mover las personas del final de un thinkLet al comienzo del siguiente.

El rol permite describir los requisitos para crear un cierto patrón de colaboración independiente de una tecnología y su configuración. Durante la ejecución de un proceso de colaboración, diferentes actores pueden asumir diferentes roles. Para cada rol, las capacidades, reglas y aún los parámetros en el thinkLet pueden cambiar generando un nuevo rol. El rol está formado por capacidades, acciones, reglas y parámetros.

Las capacidades son ciertos requisitos y privilegios con respecto al espacio de trabajo del grupo que deben ser definidos. Estos requisitos y privilegios son ofrecidos por una tecnología particular, en otras palabras, las capacidades representan los medios ofrecidos a los participantes para desarrollar exitosamente una actividad.

El thinkLet especifica un conjunto de acciones individuales para ser ejecutadas por los miembros de un grupo, creando un patrón de colaboración. Un thinkLet describe cómo efectuar las acciones específicas para cada uno de los roles en tres maneras: 1) describe las capacidades tecnológicas necesarias para que los participantes realicen las acciones requeridas; 2) describe las reglas utilizadas para condicionar las acciones individuales, y 3) describe los parámetros requeridos para instanciar la ejecución de las acciones de los participantes.

Para ejecutar un thinkLet determinado, el facilitador debe establecer las reglas que conducen a cada participante a realizar las acciones deseadas. Si cada participante usa las capacidades provistas y respeta las

reglas que el facilitador plantea, el participante debería producir las acciones esperadas.

Los parámetros representan información específica que debe ser incluida de tal manera que el thinkLet sea ejecutado significativamente.

Un thinkLet tiene asociado un thinkLetDocument, que contiene información descriptiva acerca del thinkLet, donde se incluye el nombre, resumen, propósito, explicación y ejemplo.

El nombre es una palabra o frase descriptiva que tiene relación con el patrón de colaboración que encapsula el thinkLet. El resumen da una breve descripción de los eventos y salidas del thinkLet. El propósito establece los criterios utilizados para la selección de un thinkLet. Este atributo provee información suficiente para expresar el patrón que el thinkLet produce y distinguirlo de otros thinkLets que pueden crear el mismo patrón. La explicación contiene observaciones útiles acerca de la naturaleza del thinkLet , cómo y cuando utilizarlo, recomendaciones y orientaciones basadas en experiencias de campo. El ejemplo es una narración de una experiencia previa del uso del thinkLet en un contexto de un proceso grupal que le ayuda al diseñador a entender la utilidad del thinkLet.

El composedThinkLet es un thinkLet especial que agrupa varios thinklets para formar un proceso colaborativo, la secuencia en la cual los thinkLets se ejecutan está determinada por uno o varios thinkLetTransitions, éstos son referenciados a partir de una lista ordenada de thinkLetTransitions. El composedThinkLet puede ser visto como un subproceso que puede ser instanciado dentro de un proceso más complejo.

4. Modelo de Componentes ThinkLets: ThinkLet CM

Los Componentes son unidades software reusables que pueden ser desarrollados, adquiridos e incorporados a un sistema de forma independiente en tiempo y espacio [12]. Los componentes software requieren información de especificación para usuarios y desarrolladores, que permite establecer si un componente cumple o no con los requisitos establecidos por un nuevo sistema. Los componentes tienen tres elementos de especificación: interfaces, comportamiento y propiedades.

El modelo de componentes define la arquitectura básica de componentes, especificando la estructura de las interfaces, las interacciones entre componentes y las interacciones entre componentes y el framework de soporte. La implementación del ThinkLet CM debe desplegarse en un Servidor de Aplicaciones Colaborativas (CAS). Un componente thinkLet está compuesto por una especificación que define las reglas de interacción entre el componente y el CAS. El componente thinkLet en tiempo de diseño debe ser adaptado a las interfaces de las especificaciones ThinkLet CM. El ThinkLet CM define las clases de componentes, sus interfaces y un patrón de interacción entre las clases de componentes, esto se representa en la figura 2. Este modelo tiene cuatro clases de componentes: ThinkLet, ComposedThinkLet, ThinkLetTransition y Role, que encapsulan la funcionalidad e información de los elementos del modelo conceptual.

Los componentes implementa una o más interfaces, esto le permite al componente realizar sus obligaciones y reglas, asegurando una interacción predecible en ambientes estandarizados. ThinkLet CM establece las siguientes interfaces:

- ThinkLetInterface
- ComposedThinkLetInterface
- ThinkLetTransitionInterface
- RoleInterface

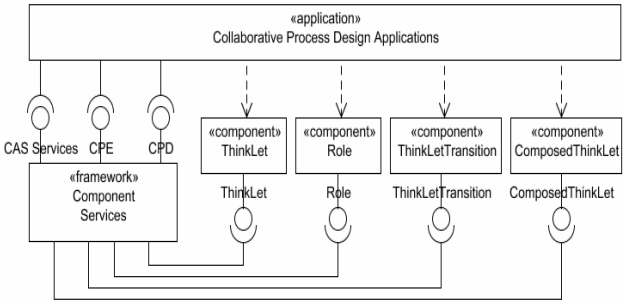


Figura 2. Modelo de componentes thinkLet: ThinkLet CM

El framework es una línea base tecnológica que ofrece un conjunto de servicios de tiempo de ejecución para soportar el ThinkLet CM. El framework gestiona los recursos compartidos por los componentes y ofrece mecanismos subyacentes para la comunicación e interacción. Los servicios básicos, llamados CAS Services, incluyen funcionalidades de comunicación, directorio, persistencia y seguridad, entre otros. Adicionalmente, el framework asiste a los desarrolladores para crear aplicaciones en el dominio de la ingeniería de la colaboración así como para su mantenimiento. También, ofrece una infraestructura bien diseñada para adicionar nuevos artefactos software en el futuro sin que se presenten inconvenientes. Las aplicaciones de diseño de procesos de colaboración son construidas sobre el framework, ampliando sus capacidades para satisfacer los requisitos de las aplicaciones.

Los servicios de dominio: Collaborative Process Design (CPD) y Collaborative Process Execution (CPE) permiten la definición y ejecución de nuevos procesos de colaboración basados en los componentes thinkLet.

Para especificar la interacción entre los componentes y por ende crear aplicaciones de diseño de procesos colaborativos es necesario establecer un patrón de interacción, de acuerdo a las cuatro clases de componentes identificadas en el ThinkLet CM, esto se ilustra en la figura 3. Las interfaces representan la funcionalidad ofrecida por los componentes, y poseen las características de las interfaces definidas en el ThinkLet CM.

Las interfaces ThinkLet, ThinkLetTransition, ThinkLetComposed, Role y RoleHome definen los métodos que permiten a un componente cliente crear, hallar y remover instancias de los componentes, así como los métodos de negocio que implementan los componentes. Estas interfaces proveen la vista cliente de los componentes.

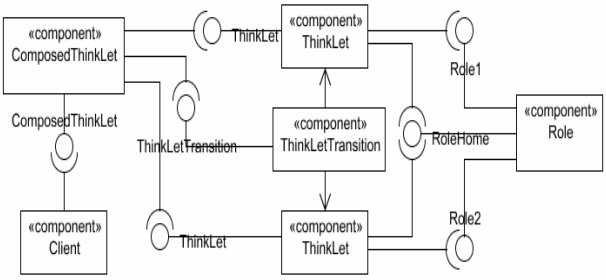


Figura 3. Patrón de interacción entre componentes

5. Discusión

En esta sección se analizan las ventajas del modelo de componentes thinklet propuesto a partir de un escenario donde se aplican los thinklets en la ingeniería de requisitos en el proceso de desarrollo software.

La ingeniería de requisitos es una de las áreas del proceso software más determinantes para la calidad de un producto software y por tanto delicada. Además es un proceso complejo debido a los diferentes participantes involucrados en la producción de los requisitos. En la práctica, todos los participantes de un sistema tienen necesidades y expectativas, que representan requisitos del sistema. Los participantes del sistema son los roles jugados por las personas o sistemas involucrados o afectados por el desarrollo del nuevo sistema. Estos incluyen los ejecutivos, usuarios finales, analistas de mercado, gestores técnicos y desarrolladores, entre otros. Para que la ingeniería de requisitos sea exitosa debe identificarse una comunidad apropiada de participantes [13].

Para conciliar las diversas necesidades y expectativas de los participantes es necesario encontrar un equilibrio, para lo cual se requiere de mecanismos para la captura y análisis de los diferentes requisitos de los participantes, identificar los conflictos entre requisitos, tomar decisiones en los casos de conflicto y registrar los resultados de estas decisiones.

La ingeniería de requisitos puede ser mejorada utilizando una estrategia colaborativa basada en thinklets, produciendo requisitos más apropiados que conduzcan a mejores sistemas. La ingeniería de requisitos comprende la captura, especificación y validación de requisitos [14]. Los casos de uso son una práctica muy utilizada en la captura de requisitos dentro del proceso software. Esta práctica puede beneficiarse del diseño y despliegue de procesos de colaboración, para ilustrar esto se plantea la utilización del modelo de componentes thinklets propuesto en el diseño del proceso de captura de requisitos.

Considere la actividad inicial en el proceso de identificación de los casos de uso, donde se necesita identificar los actores y sus acciones. Un actor caracteriza un usuario externo o conjunto de usuarios relacionados que interactúan con el sistema [15].

En el modelo de casos de uso, los actores son las únicas entidades externas que interactúan con el sistema. Existen variaciones en cómo los actores son modelados [16]. Un actor es frecuentemente un usuario humano, en muchos sistemas los humanos son los únicos actores, sin embargo es posible para algunos sistemas que un actor sea un sistema externo. En algunas aplicaciones, un actor también puede ser un dispositivo externo de Entrada/Salida o un temporizador. Estos actores son particularmente predominantes en sistemas empotrados de tiempo real, en los cuales el sistema interactúa con el ambiente externo a través de sensores y actuadores [14].

El subproceso de identificación de actores realizado por los participantes, en el esfuerzo de captura de requisitos, puede ser modelado usando thinkLets como se representa en la figura 4.

El subproceso incluye cinco thinkLets y un punto de decisión. Los thinkLets usados son: DirectedBrainstorm (Divergencia), FastFocus (Convergencia), PopcornSort (Organización), BucketWalk (Evaluación) y MakeContract (Construir consenso).

En el thinkLet DirectedBrainstorm, los participantes generan ideas como respuesta a una sola pregunta acerca de los potenciales actores y las acciones iniciadas por ellos. Los participantes generan nuevas ideas o comentan las ideas previas. Esto a menudo genera una larga lista de ideas, muchas de las cuales pueden ser redundantes, irrelevantes, ambiguas o tener diferentes niveles de abstracción.

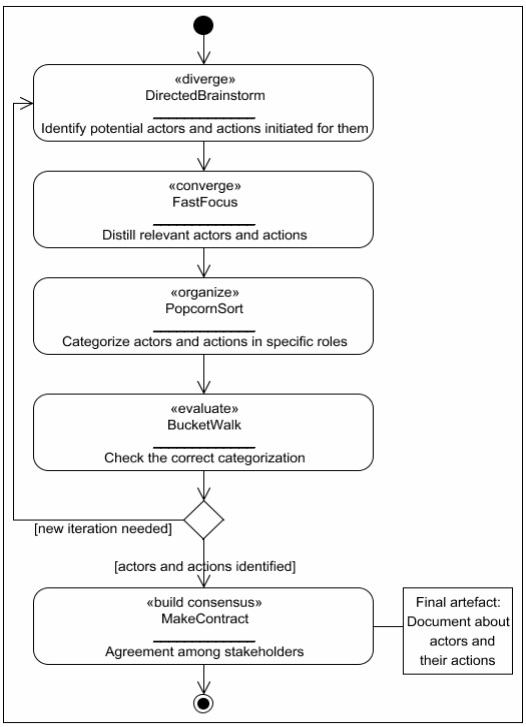


Figura 4. Diagrama del proceso de identificación de actores

El thinkLet FastFocus es utilizado para depurar esta lista, los participantes examinan las contribuciones de la lluvia de ideas y proponen en voz alta los aspectos que deben ser considerados como claves discutiendo su significado y redacción. Cuando los participantes no encuentren más aspectos claves la actividad termina generando una lista de aspectos claves libre de redundancias. Después del thinklet FastFocus es necesario categorizar los actores y sus acciones, el thinkLet PopcornSort es utilizado para categorizar la lista de actores, en él los participantes deben clasificar las diferentes ideas dentro de categorías establecidas. El siguiente paso en este subproceso es la evaluación de la lista categorizada de actores y acciones, el thinklet BucketWalk sirve para verificar la correcta categorización. En esta parte, se debe decidir si se identificaron todos los actores o si es necesaria otra iteración. Finalmente, el thinklet MakeContract es utilizado para crear un acuerdo entre los participantes y obtener el documento final con los actores y acciones.

La figura 5 presenta el proceso de identificación de casos de uso en una vista de componentes, esta es utilizada para especificar los thinklets implementados como componentes thinklet de acuerdo al Thinklet CM.

La gráfica muestra el componente UseCasesIdentification, el cual representa el proceso de identificación de casos de uso, este componente ofrece la interfaz UCI mediante la cual un componente de presentación cliente puede configurar, instanciar y ejecutar el proceso de colaboración. Se relaciona con componentes thinkLet y thinkLetTransition mediante las interfaces requeridas.

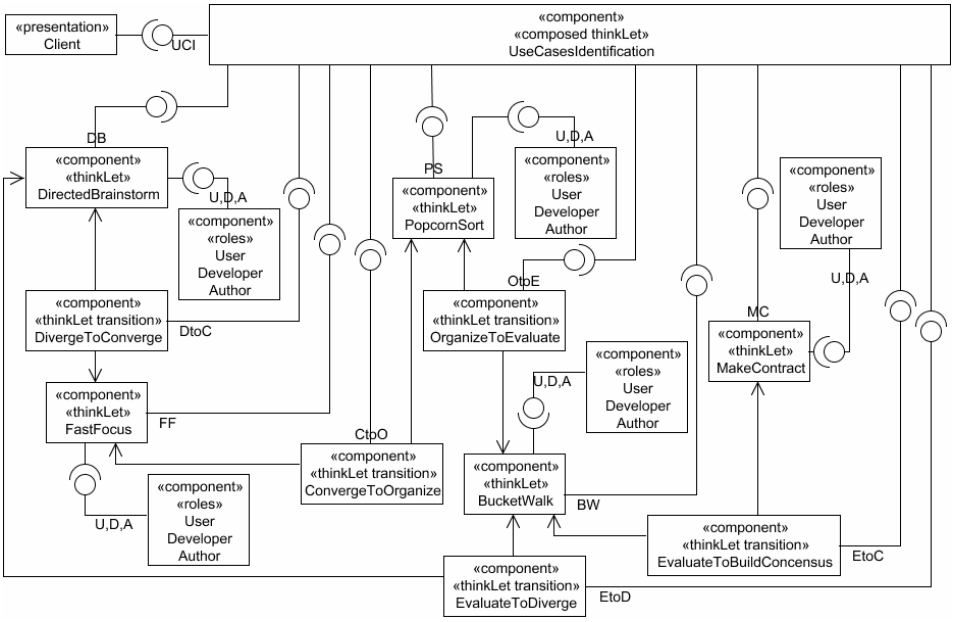


Figura 5. Proceso de identificación de casos de uso en una vista de componentes

El componente thikLetTransition referencia parejas de componentes thinkLets para establecer la secuencia de ejecución del proceso de colaboración, con el fin de moverse a través de los patrones de colaboración. El componente Role permite definir las responsabilidades y capacidades de los actores que intervienen en el proceso, en términos de las reglas establecidas para el tipo de thinkLet al cual esta asociado el rol.

En la figura 6 se ilustra la interacción entre instancias de componentes que se da en el diseño del proceso de identificación de casos de uso. Se presentan dos instancias de componentes thinkLets (DirectedBrainstorm y FastFocus), una instancia del componente thinkLetComposed (UseCasesIdentification), el componente Role y una instancia del componente thinkLetTransition (DivergeToConverge). La colaboración entre los componentes se realiza a partir de los mensajes intercambiados entre ellos.

El usuario de las aplicaciones de diseño de procesos de colaboración accede a las capacidades ofrecidas por los componentes thinkLets a partir de componentes cliente, como se muestra en la figura 6, los cuales son componentes de presentación que se implementan a partir de tecnologías relacionadas tales como JSF o JSP. Los componentes de presentación no se especifican en el ThinkLet CM dándole flexibilidad al proveedor de la herramienta de diseño seleccionar la tecnología más apropiada y adaptar el desarrollo del sistema a metas de negocio particulares.

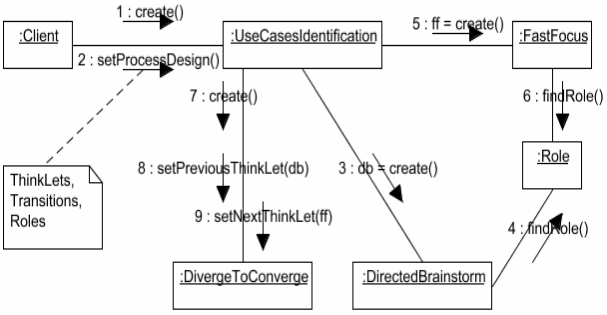


Figura 6. Diagrama de colaboración proceso de identificación de casos de uso

6. Conclusiones y Trabajo Futuro

Es posible conducir los esfuerzos de diseño y ejecución de procesos colaborativos de los grupos de trabajo, mediante el uso del modelo conceptual de thinkLets y el modelo de componentes thinkLet. Esto se ilustró mediante un escenario de aplicación de estos modelos en la práctica de identificación de casos de uso, en la ingeniería de requisitos en un proceso de desarrollo de software. La inclusión de este enfoque colaborativo a partir de estos modelos contribuye a la mejora de los procesos y por lo tanto a la productividad de las organizaciones.

El concepto de thinkLet y la definición de un modelo conceptual, pueden ser mapeados a componentes software de tal manera que sea posible desarrollar una infraestructura tecnológica soportada sobre una especificación e implementación estándar de componentes. La implementación ofrece un conjunto de servicios subyacentes que favorecen la comunicación e interacción de los componentes, y garantizan que las aplicaciones desarrolladas cumplan con los requisitos de atributos de calidad como desempeño, disponibilidad, seguridad y modificabilidad.

Para la especificación de los procesos de colaboración se utilizo el lenguaje de modelado unificado (UML 2.0), el cual permitió describir los thinklets, los componentes thinklet y los diseños de los procesos colaborativos de forma precisa y clara. Los diagramas de UML utilizados para tal fin son el diagrama de clases, componentes y actividades. Estos medios pueden ser considerados como mecanismos estándar para modelar y comunicar procesos colaborativos.

Como trabajo futuro se debe definir y modelar otras áreas del proceso de desarrollo de software usando thinklets y soportarlo sobre implementaciones basadas en el thinkLet CM, este enfoque puede ser adoptado por las organizaciones de software como una estrategia de mejora de sus procesos.

El siguiente paso es implementar el framework desplegando el modelo de componentes thinkLet en un servidor de aplicaciones colaborativas, este puede ser configurado sobre un servidor de aplicaciones J2EE donde los componentes de entidad pueden ser usados para soportar los componentes Role, los componentes de sesión pueden ser usados para crear la lógica de las aplicaciones de colaboración, es decir los componentes ThinkLet, ThinkLetTransition y ComposedthinkLet, y los componentes Web para ofrecer las capacidades de presentación mediante interfaces gráficas de usuario.

Dado que los componentes thinkLet son las unidades básicas para la construcción de aplicaciones de diseño y despliegue de los procesos de colaboración, es necesario gestionarlos mediante un repositorio donde sea posible realizar operaciones de control de versiones y gestión de la configuración. Esto facilitará el desarrollo conjunto, el mantenimiento y actualización de las aplicaciones de colaboración.

Agradecimientos

Este trabajo ha sido realizado en el marco de los Proyectos: COMPETISOFT, financiado por el Programa Iberoamericano de Ciencia y Tecnología para el Desarrollo – CYTED; Programa Jóvenes Investigadores e Innovadores 2005 Contrato 051 de 2005, financiado por Colciencias y la Universidad del Cauca, y NIC Chile a través de la Beca NIC Chile 2006 del Departamento de Ciencias de la Computación – Universidad de Chile.

7. Referencias

[1] Kolfschoten, G., Briggs, R., Appelman, J., and Vreede, G., "Thinklets as Building Blocks for Collaboration Processes: A further Conceptualization", *CRIWG 2004*, Springer, 2004, pp.137-152.

[2] Nunamaker, J. Jr., Briggs, R., Vreede, G., "From Information Technology to Value Creation Technology", *Information Technology and the Future Enterprise*, Prentice-Hall, New York, 2001.

[3] Vreede, G., Briggs, R., "Collaboration Engineering: Designing Repeatable Processes for High-Value Collaborative Tasks", *Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05)*, IEEE Computer Society Press, Los Alamitos, 2005.

[4] Briggs, R., Vreede, G., Nunamaker, J., David, T., "Thinklets: Achieving Predictable, Repeatable Patterns of Group Interaction with Group Support Systems (GSS)", *Proceedings of the 34th Hawaii International Conference on System Sciences*, IEEE Computer Society Press, Los Alamitos, 2001.

[5] Kolfschoten, G., Appelman, J., Briggs, R., Vreede, G., "Recurring Patterns of Facilitation Interventions in GSS Sessions", *Proceedings of the 37th Hawaiian Internal Conference on System Sciences*, IEEE Computer Society Press, Los Alamitos, 2004.

[6] Li, D., Yang, Y., Creel, J., Dworaczyk, B., "A Framework for Building Collaboration Tools by Leveraging Industrial Components", *CoopIS'06*, Montpellier-France, 2006.

[7] Dewan, P., "Architectures for Collaborative Applications", *CSCW*, John Wiley and Sons, New York, 1999, Chapter 7.

[8] Roussev, V., Dewan, P., Jain, V., "Composable Collaboration Infrastructures Based on Programming Patterns", *ACM CSCW Conference*, 2000, pp. 117–126.

[9] Marsic, I., "An Architecture for Heterogeneous Groupware Application", *Proc. ICSE' 01*, IEEE, 2001, pp. 475-484.

[10] Guicking, A., Grasse, T., "A Framework Designed for Synchronous Groupware Applications in Heterogeneous Environments", *12th International Workshop CRIWG 2006*, Springer, Medina del Campo-Spain, 2006, pp.203-218.

[11] Collazos, C., Hermida, V., Hurtado, J., Tobar, C., "Toward a Software Product Line for Collaborative Process Design Based on ThinkLets", *Euro American Conference on Telematics and Information Systems EATIS 2006*, Santa Marta-Colombia, 2006, ISBN: 958-8166-38-1, pp.182-190.

[12] Szyspersky, C., Pfister, C., "Component-Oriented Programming". *Special Issues in Object Oriented Programming–ECOOP96 Workshop Reader*, Dpunkt Verlag Heiderberg, 1997.

[13] Clements, P., Northrop, L. *Software Product Lines: Practices and Patterns*, Addison Wesley, Boston, 2002.

[14] Gomaa, H., *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures*, Addison-Wesley, Boston, 2005.

[15] Rumbaugh, J., Booch, G., Jacobson, I., *The Unified Modeling Language Reference Manual*, Addison-Wesley 2nd ed., Boston, 2005.

[16] Fowler, M., *UML Distilled: Applying the Standard Object Modeling Language*, Addison-Wesley 3rd ed., Boston, 2004.

Monitoreo del Desempeño de los Factores de Seguridad de una Transacción Web a través de la Interfaz de Usuario

R. Mendoza González[φ], J. Muñoz Arteaga[φ], F. J. Álvarez Rodríguez[φ], y M. Vargas Martin[α]
[φ] *Universidad Autónoma de Aguascalientes, Aguascalientes, Ags. México.*
mendozagric@yahoo.com.mx; {jmunozar, fjalvar}@correo.uaa.mx
[α] *University of Ontario Institute of Technology, Oshawa, Canada.*
miguel.vargasmartin@uoit.ca

Resumen

Es muy importante que la interfaz de una aplicación o servicio Web tenga un buen diseño y mas aún si de ese diseño depende la efectividad de una herramienta específica de seguridad. En la actualidad existen varios conceptos de diseño para lograr lo anterior tales como la Interacción Humano Computadora (IHC) o la nueva ampliación de la misma, la llamada IHC-S, la cual esta enfocada a interfaces de usuario de aplicaciones específicas de seguridad, sin embargo, estos conceptos y criterios no son utilizados por los diseñadores o programadores al momento de generar la interfaz de una herramienta de seguridad. En este artículo proponemos una interfaz para notificar, al usuario o solicitante, sobre el desempeño de los factores de seguridad y alertarlo si una amenaza es detectada a partir de dicho desempeño; esto, aplicando los criterios de la IHC-S para mantener la usabilidad de dicha interfaz.

1. Introducción

En la actualidad, cada vez mas usuarios han tenido contacto con los elementos de una interfaz de usuario de alguna tecnología, tales como menús, botones, hyperlinks en Internet, iconos y ventanas de algún sistema operativo, entre otras, los cuales, tienen como objetivo permitir al usuario un manejo práctico y entendible de las diferentes tecnologías que utilice, lo cual se puede interpretar como: "*Hacer sentir al usuario que tiene el control de una determinada tecnología"* [1].

El aumento en el número de computadoras en oficinas y casas provocó que muchos usuarios sin experiencia en aspectos concernientes a la seguridad de la información se vieran obligados a interactuar con varias tecnologías especializadas en dicho tópico, las cuales requieren, para un buen manejo de las mismas, las siguientes características:

- Una interfaz adecuada, que transmita y guíe al usuario a través de las características específicas de seguridad de dicha tecnología.
- La interfaz debe transmitir las experiencias de seguridad funcionalmente por medio de la interfaz.
- La interfaz debe avisar al usuario sobre las opciones de seguridad disponibles y cómo usarlas.
- La interfaz debe hacer que el usuario no ignore las características de seguridad de una tecnología y de que no las use incorrectamente. Por ejemplo, una determinada tecnología para la seguridad de la información puede proteger la información solo si esta activada, y solo puede ser activada si el usuario sabe como hacerlo. La interfaz debe asegurar que el usuario será guiado para realizar dicha tarea.

La mayoría de las percepciones del usuario ante una tecnología se basan en sus experiencias con las interfaces, este campo de investigación le concierne a la IHC (Interacción Humano-Computadora), la cual, desde el punto de vista de las Ciencias Computacionales "*trata con la interacción entre uno o mas humanos y una o mas computadoras usando la interfaz de un programa*" [2]. Los conceptos probados de la Interacción Humano-Computadora pueden ser usados tanto para diseñar una interfaz como para mejorar una ya existente, considerado aspectos como la usabilidad, la cual determina el grado de facilidad de uso de una determinada tecnología, el grado eficiente de respuesta de dicha tecnología ante las necesidades de quien la utiliza, y la satisfacción del usuario con los resultados obtenidos por el uso de una tecnología mediante la realización de sus tareas especificas [3].

Existe una ampliación a la IHC, llamada Interacción Humano Computadora para aplicaciones específicas de Seguridad (IHC-S), la cual modifica, condensa y adapta los conceptos de la Interacción Humano-Computadora para enfocarse a aspectos de seguridad y

busca el mejoramiento de la misma mediante la interfaz. Hasta el momento no se encuentra en la literatura una referencia estándar sobre este nuevo término por lo que para este artículo se utilizará la definición propuesta por Jonhston y otros en [1] la cual textualmente define a la IHC-S de la siguiente manera: "*The part of a user interface which is responsible for establishing the common ground between a user and the security features of a system. HCI-S is human computer interaction applied in the area of computer security*". Con base en esta definición se puede decir que la Interacción Humano-Computadora para aplicaciones específicas de Seguridad, es la parte de una interfaz de usuario que es responsable del establecimiento de una apropiada interacción entre un usuario y las características de seguridad de un sistema. Como Johnston y otros afirman, la IHC-S trata sobre como las características de seguridad de una interfaz gráfica sean lo mas amigables e intuitivas posible para el usuario, ya que mientras mas fácil de usar sea un sistema, menos posibilidades habrá de que el usuario cometa un error o ignore alguna característica de seguridad, llevando con esto a que un sistema sea más seguro y confiable [1].

2. Problemática

Es importante que el usuario o solicitante de un servicio Web este consiente del desempeño de los factores de seguridad por parte de los elementos del esquema de seguridad para las transacciones que se utilice, lo cual puede lograrse mediante su monitoreo a través de los elementos de la interfaz de usuario (IU), debiéndose considerar los siguientes puntos:

- Es necesario que el cliente o solicitante del servicio Web tenga certeza de la seguridad existente durante la transacción que se este realizando, y, en caso de que esta no sea segura, indicarle las acciones que se deben realiza para evitar o corregir algún error provocado por una amenaza.
- La presentación de las acciones que se deben tomar en caso de verse comprometida la seguridad y de las características de seguridad del servicio Web a través de la interfaz gráfica debe ser de tal manera que sean percibidas por el usuario como fáciles de realizar y utilizar.
- Para facilitar el uso de las características de seguridad y la realización de las acciones para evitar y corregir errores se deben considerar los nuevos criterios de diseño de la Interacción Humano-Computadora para aplicaciones específicas de Seguridad, los cuales han sido ignorados por la mayoría de las tecnologías de seguridad.

Al tomar en cuenta los puntos anteriores y aplicarlos se beneficiaria en gran medida al usuario o solicitante ya que sería alertado de una manera adecuada y conveniente al detectarse algún incumplimiento de los factores de seguridad establecidos, haciéndole saber de las características de seguridad existentes y guiándolo para el adecuado uso de las mismas mediante los componentes de la interfaz para mantener la integridad y la confidencialidad de la información durante una transacción entre un solicitante y un servicio Web.

3. Criterios de Interacción Humano-Computadora aplicaciones específicas en Seguridad

Para una exitosa implementación de la Interacción Humano-Computadora en aplicaciones específicas de Seguridad se deben considerar los criterios propuestos en, los cuales, ayudarán en el diseño y desarrollo de las interfaces usadas dentro de un ambiente de seguridad, dichos criterios están basados en los diez principios generales de IHC para el diseño de interfaces de usuario presentados en [6] y fueron modificados para ser usados esencialmente en ambientes de seguridad. Los seis criterios de la Interacción Humano-Computadora para aplicaciones específicas de Seguridad son:

1. **Visibilidad del estado del sistema:** La interfaz debe permitir al usuario o solicitante observar el estado interno del sistema, por ejemplo mediante un mensaje que le indique que una determinada característica de seguridad del sistema esta activada. Los mensajes de error como el de "Fallo de la transacción" deben ser detallados pero específicos, deben incluir la acción que se necesita tomar a causa del error y como se puede obtener una asistencia adicional. "Los mensajes genéricos de error no son adecuados" [1].
2. **Diseño estético y minimalista de la interfaz:** La información presentada al usuario o solicitante debe ser suficiente para usuarios con poca experiencia pero al mismo tiempo no debe ser demasiada para usuarios experimentados, esto haciendo un balance en cuanto a la información presentada no incluyendo información irrelevante. El usuario o solicitante no debe ser bombardeado con información y opciones evitando en lo posible el uso de términos técnicos, por último, la interfaz no debe lucir complicada o confusa, debe mantenerse siempre un diseño minimalista.
3. **Satisfacción:** Las actividades de seguridad deben ser fáciles de realizar y no deben presentarse como un tema técnico al usuario o solicitante, en algunos casos es conveniente usar el humor o gráficas para la

introducción de conceptos importantes de seguridad al usuario o solicitante de una manera mas entretenida.

4. **Transmisión de características:** La interfaz debe informar al usuario o solicitante de una manera clara sobre las características de seguridad, una buena manera de hacerlo es usando gráficos y figuras.
5. **Aprendizaje:** La interfaz debe ser lo mas fácil de aprender y amigable posible, esto puede lograrse mediante el uso de metáforas del mundo real, por ejemplo, utilizar imágenes de llaves y candados cuyo significado en el mundo real puede ser transportado a las interfaces de seguridad, así el usuario o solicitante reconocerá dichos elementos y tendrá una idea de cómo usarlos. Es importante mencionar que una interfaz basada en estándares es mucho mas fácil de aprender, por ejemplo, muchos usuarios o solicitantes están familiarizados con el aspecto de la interfaz de Microsoft Windows lo que provoca que un nuevo programa cuya interfaz este basada en la de Windows (íconos, menús, ventanas, etc.) sea fácil de aprender para un usuario. Otro ejemplo es el uso de palabras comunes como "Contraseña" en lugar de "Código de Acceso".
6. **Confianza:** La exitosa aplicación de los criterios anteriores dará como resultado que el usuario o solicitante tenga confianza en un sistema o servicio Web, entendiendo por confianza, que el usuario o solicitante crea o tenga una buena disposición de creer en la seguridad de un sistema o servicio Web. Así, el grado de confianza que un usuario o solicitante tenga en un sistema o servicio Web determinará cuanto ese usuario o solicitante utilice dicho servicio, el grado de confianza puede incrementarse informando al usuario o solicitante, por medio de la interfaz haciendo uso de estos criterios, sobre los riesgos y como dichos riesgos pueden ser minimizados.

Del mismo modo, con varios de estos criterios se solapan los factores que transmiten confianza en un ambiente de comercio electrónico propuestos en [7] dichos factores son: El cumplimiento, la tecnología, los sellos de aprobación, la presentación, la navegación y la marca.

4. Monitoreo gráfico del desempeño de los factores de seguridad

Atendiendo a la problemática descrita en la sección 2 de este artículo, se presenta un caso de estudio en el cual se usa un escenario en el que se requiere una interfaz que informe al usuario o solicitante sobre el desempeño de los factores de seguridad y se le indiquen las acciones a realizar en caso de que ocurra un error causado por alguna amenaza presente durante la transacción Web, esto considerando los criterios mencionados en la sección 3 de este artículo.

Para determinar el desempeño de los factores de seguridad en el escenario propuesto se utilizaran los resultados de las métricas exploratorias propuestas en [5], los cuales, se presentarán en la interfaz de usuario.

Para la obtención de resultados de las métricas se tomaran las siguientes consideraciones presentadas en la Tabla 1.

Tabla 1. Consideraciones para la presentación de las evaluaciones.

Métrica	**Elemento del Esquema / Factor**	**Rango de Desempeño en Porcentaje**		
		Bueno	Medio	Malo
M5	Detección de Intrusos / (**Confidencialidad**)	100 - 95	94 - 90	89 - 0
M4	Encriptación de Información / (**Integridad**)	100 - 95	94 - 90	89 - 0
M1 y M2	Control de Acceso Basado en Roles / (**Disponibilidad y Auditoria**)	100 - 90	89 - 80	79 - 0
M3	Autenticación del Servicio Web / (**Autenticación**)	100 - 95	94 - 90	89 - 0

En la Tabla 1 se consideraron los siguientes factores de seguridad: Confidencialidad, Integridad, Disponibilidad, Auditoria, y Autenticación, a cada uno de los cuales se les asignó un "Rango de Desempeño". Es importante mencionar que los valores del "Rango de Desempeño", son proporcionados por el diseñador o programador del servicio Web, atendiendo a los requerimientos de seguridad específicos de una determinada organización, en la Tabla 1 se presentan los valores del "Rango de Desempeño" determinados para este escenario en particular. Así mismo, en la tabla anterior (Tabla 1) se hace una distinción entre los diferentes "Rangos de Desempeño", lo cual es clave para la presentación de los resultados de las evaluaciones en la interfaz de usuario, ya que a cada rango (Bueno, Medio, o Malo) se le asignará un color especifico, verde, amarillo y rojo, respectivamente, para ser mostrado al usuario o solicitante durante la transacción en un lugar específico de la interfaz indicándole así el nivel de seguridad. Por ejemplo, si durante la transacción se ve comprometida de alguna forma la seguridad al establecerse un bajo cumplimiento de alguno de los factores de seguridad,

se coloreará el marco de la interfaz ya sea de color amarillo o rojo dependiendo de la gravedad de la amenaza con base en el resultado obtenido a partir de la aplicación de una métrica especifica y el rango de cumplimiento asociado. En caso de que los niveles de desempeño se mantengan dentro del valor más alto, el marco de la interfaz será verde. Adicionalmente a la representación del nivel de seguridad durante una transacción por medio de colores, para este escenario específico, se mostraran en la interfaz de usuario, el porcentaje del "Rango de Desempeño" resultante así como varios elementos para alertado adecuadamente al detectarse algún incumplimiento de los factores de seguridad establecidos, haciéndole saber de las características de seguridad existentes y guiándolo para el adecuado uso de las mismas.

Tanto el esquema de seguridad como las métricas exploratorias son parte del elemento "Intermediario" del modelo de seguridad presentado en la Figura 1, en la cual se describe de manera gráfica el escenario base.

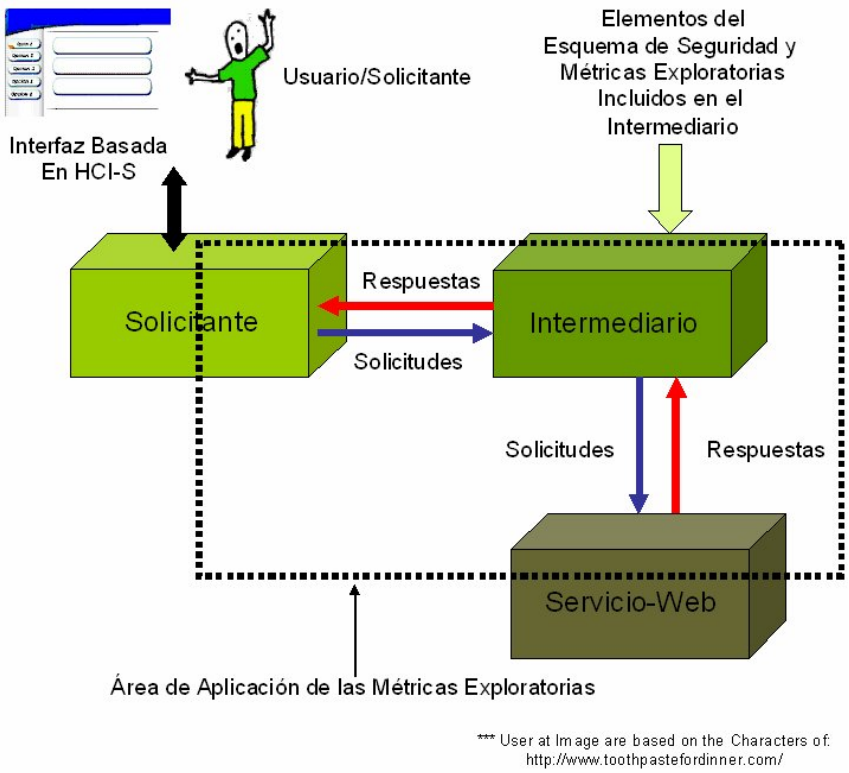


Figura 1. Escenario base

Con base en el escenario anterior, se generó una propuesta para el diseño de una interfaz la cual fue creada en Java y se describe a continuación considerando cada uno de los 6 criterios de la IHC-S propuestos por Johnston y otros en [1] y definidos en la sección 3 de este artículo.

1. **Transmisión de características:** Mediante el uso de una imagen de un semáforo y el mensaje "Módulo de transacción segura ACTIVADO", se alertará al usuario o solicitante sobre el hecho de que su transacción esta siendo protegida motivándolo así a confiar en el sistema o servicio Web.
2. **Visibilidad del estado del sistema:** Mediante el coloreo del marco de la interfaz, el cambio de color en la imagen de un semáforo incluido en la interfaz, y el despliegue de un mensaje de alerta, se le notificará al usuario o solicitante sobre el estado interno del sistema, es decir si se ha detectado alguna amenaza que pudiera comprometer la integridad o confidencialidad de la transacción. Los mensajes de notificación o alerta son específicos y concretos sin el uso de tecnicismos ni información irrelevante, así mismo, se incluyen: una recomendación de la acción que se debe realizar a causa del error y una opción para obtener una información y asistencia adicional.
3. **Aprendizaje:** La interfaz es fácil de aprender y amigable, ya que se le informa textualmente al usuario o solicitante que su transacción esta siendo

protegida y en caso de detectarse una amenaza se le informa de una manera fácil de interpretar y entender mediante el uso de colores y mensajes concretos y específicos.

4. **Diseño estético y minimalista de la interfaz:** La información presentada en el mensaje de alerta es concreta y específica y notificación al usuario o solicitante sobre la activación de la seguridad y de la detección de alguna amenaza se realiza de manera simple manteniéndose, en la interfaz, un minimalista y estético.
5. **Satisfacción:** Al proporcionarle al usuario o solicitante facilidad para realizar ciertas actividades de seguridad sin la inclusión de tecnicismos y el hecho de saber que su transacción esta siendo protegida y que en caso de detectarse una amenaza se le notificará y tendrá la opción de decidir si continua o no con la transacción, se sentirá satisfecho con el sistema o servicio Web y con la seguridad del mismo.
6. **Confianza:** El hecho de mantener al usuario o solicitante consiente de que su transacción esta siendo protegida, e informarlo en caso de detectarse una amenaza sobre las acciones a tomar para evitarla, se traduce en confianza, por parte del usuario o solicitante, hacia el sistema o servicio Web.

A continuación se presenta, gráficamente, la interfaz de usuario descrita mediante los criterios anteriores.

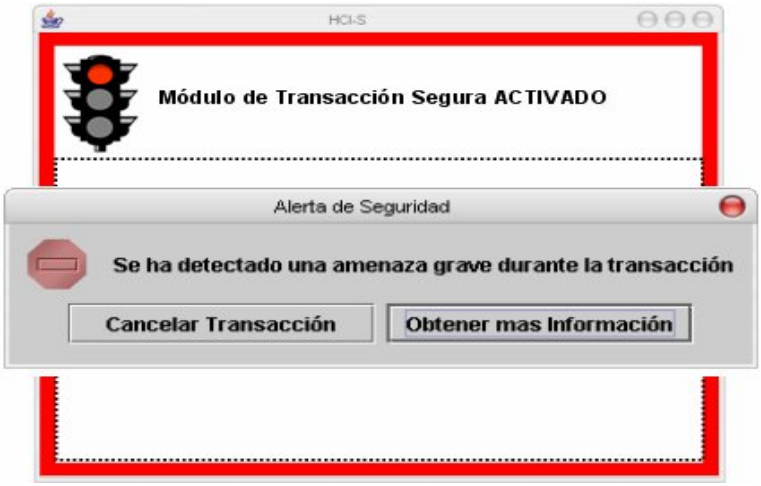


Figura 2. Notificación al usuario o solicitante sobre una amenaza de alto riesgo.

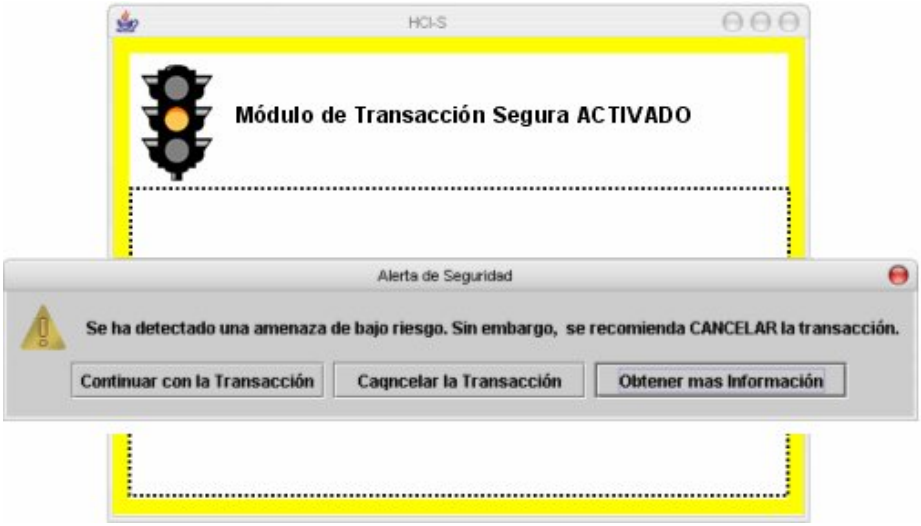


Figura 3. Notificación al usuario o solicitante sobre una amenaza de bajo riesgo.

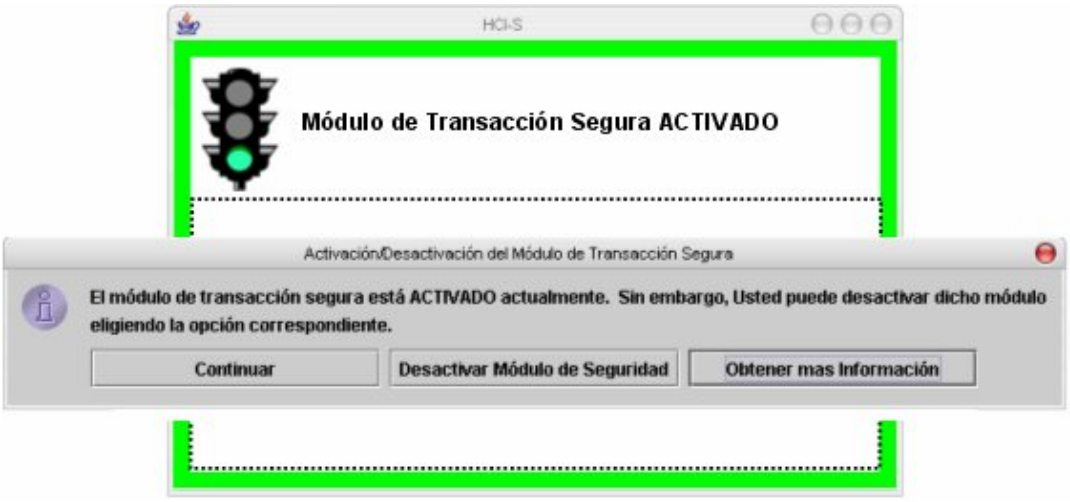


Figura 4. Notificación al usuario o solicitante sobre una transacción segura.

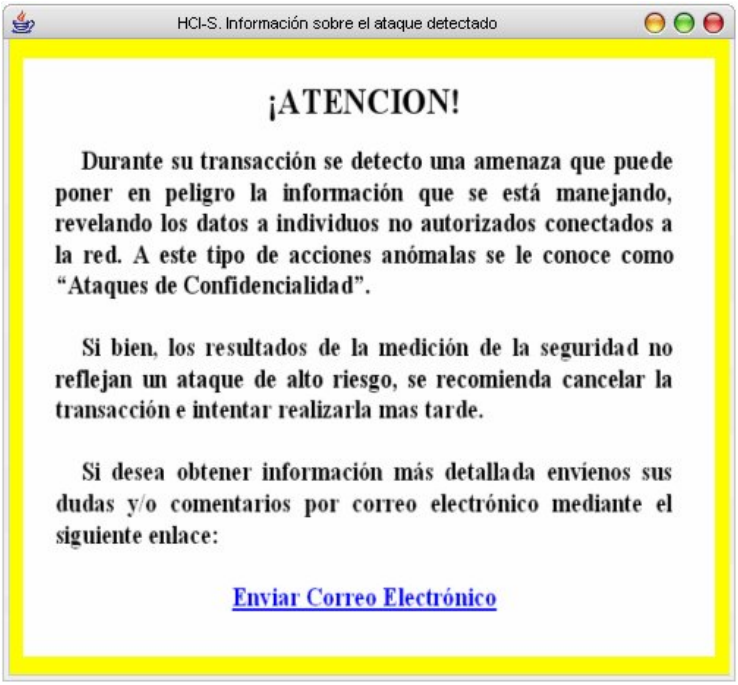


Figura 5. Despliegue de información adicional al usuario o solicitante.

En la Figura 2 se puede observar que al detectarse una amenaza de alto riesgo se le hace saber al usuario mediante el uso del color rojo, color típico de alarma, tanto en el marco de la interfaz como en el semáforo incluido en la misma, y con el despliegue de un mensaje que le indica la acción que se debe realizar ante dicha amenaza y un enlace para obtener mas información sobre dicho ataque y asistencia adicional.

Similarmente, en la Figura 3, se informa al usuario sobre la detección de una amenaza pero en este caso no tan seria como la primera, por lo que se usa tanto en el marco como en el semáforo el color amarillo, frecuentemente utilizado en el mundo real como alerta, del mismo modo se despliega un mensaje con la característica de que se le da al usuario una sugerencia de la acción a seguir, pudiendo decidir entre continuar o no con la transacción, esto debido al bajo riesgo de la amenaza, se incluye también un enlace para obtener información adicional sobre la amenaza y asistencia adicional.

En la Figura 4, se usa el color verde tanto en el marco como en el semáforo para indicarle al usuario

que la transacción es segura, siendo esta la apariencia de la interfaz de usuario al momento de iniciar una transacción. Cabe mencionar que el texto "Módulo de Transacción Segura ACTIVADO" se presenta en todo momento para informar al usuario o solicitante que su transacción esta siendo protegida.

Así mismo, al iniciar la transacción se presenta al usuario o solicitante, la opción de desactivar el módulo de seguridad o continuar la transacción haciendo uso del mismo, otorgándole una mayor sensación de control.

Por último, en la figura 5 se presenta el despliegue de información específica, en este caso, sobre una amenaza de bajo riesgo, al usuario o solicitante después de haber sido seleccionada la opción "Obtener mas Información" en la caja de diálogo que aparece al detectarse dicha amenaza (ver figura 3), la información es presentada de una manera fácil de interpretar y sin hacer uso de muchos tecnicismos, se incluyen el tipo de ataque y la manera en la que podría afectar a la información, así mismo, la pantalla contiene un enlace para obtener información mas detallada o técnica vía correo electrónico. Una pantalla similar se despliega al seleccionar la opción "Obtener mas Información" en cualquiera de las cajas de diálogo que la incluyen (ver figuras 3 y 4), la información específica de cada una de las opciones es presentada al usuario o solicitante considerando las características anteriores.

La línea punteada en las figuras anteriores (ver figuras 2, 3, y 4), representa el área de la interfaz específica de un servicio Web en particular, siendo considerados únicamente los elementos de la interfaz a través de los cuales se informará al usuario o solicitante sobre el desempeño de la seguridad durante una transacción, ya que dichos elementos son los que corresponden a la investigación presentada este artículo.

5. Trabajos relacionados

En esta sección se presentan algunos de los trabajos relacionados más significativos con el fin de realizar una breve comparación entre estos y la propuesta reportada en este artículo, considerando los siguientes criterios de comparación:

1. La utilización de los criterios de la IHC-S para el diseño de interfaces.
2. Consideración de la usabilidad en una interfaz propuesta.
3. La aplicación de la interfaz propuesta en un escenario específico.

Los trabajos considerados hasta el momento para ser comparados son:

1. La investigación realizada por Johnston y otros presentada en [1].
2. El trabajo de investigación de García Ruiz y otros presentada en [8].
3. La propuesta hecha por Hewett y otros presentada en [2].

Esta información se presenta en la siguiente tabla (Tabla 2).

Tabla 2. Comparación de los trabajos relacionados.

Criterios / Trabajos	Utilización de los criterios de la IHC-S	Usabilidad en un interfaz propuesta	Aplicación de la interfaz propuesta
Johnston y otros [1]	**X**	**X**	**X**
García Ruiz y otros [8]			**X**
Hewett y otros [2]		**X**	**X**

En la Tabla 2 se presenta el cumplimiento, de los criterios planteados, por parte de los trabajos de investigación relacionados. En lo que respecta al trabajo efectuado por García Ruiz y otros presentado en [8] se define una interfaz muy completa pensada para ser usada en una herramienta para la detección de intrusos, pero en su metodología y planteamiento teórico luce muy técnica y si bien, los autores toman en cuenta varios conceptos de la Interacción Humano-Computadora, no consideran los criterios de la IHC-S teniendo con esto poca usabilidad. Dentro de los aspectos a considerar como trabajo futuro presentado por los autores, pretenden implementar su interfaz en una herramienta específica para el manejo de información bio-molecular.

La investigación desarrollada por Hewett y otros presentada en [2] propone una excelente y muy completa descripción de la Interacción Humano-Computadora haciendo uso de definiciones, conceptos y metodologías, incluso propone varios casos de estudio para ejemplificar la aplicación de la Interacción Humano-Computadora en el diseño de interfaces. Pero no consideran los criterios específicos para el diseño de interfaces de aplicaciones de seguridad de la IHC-S. Por último, la investigación realizada por Johnston y otros presentada en [1], cumple con los tres criterios especificados en la Tabla 2, pero solo analiza y hace ciertas recomendaciones a una interfaz de una aplicación de seguridad ya existente.

En la investigación presentada en este artículo, se cumple también con los tres criterios pero, a diferencia del trabajo de presentado en [1], nosotros realizamos una propuesta de una nueva interfaz generada aplicando los criterios de la IHC-S para presentar al

usuario adecuadamente información específica de seguridad.

6. Conclusiones y trabajo futuro

En este artículo se propone una interfaz basada en los criterios de IHC-S. Dicha interfaz esta orientada a incrementar la seguridad durante una transacción, al ser presentado al usuario el desempeño de los factores de seguridad después de ser evaluados por un conjunto de métricas exploratorias. Así mismo, el diseño de la interfaz permite que se muestren las características de seguridad de un sistema o servicio Web, manteniendo informado convenientemente al usuario o solicitante sobre la detección de amenazas y sobre las acciones que se deben realizar.

La interfaz propuesta en este artículo es fácil de interpretar, ya que el uso de sus elementos está basado en el reconocimiento y no en la memorización, lo cual permite una mejor utilización de las características de seguridad por parte del usuario o solicitante, manteniéndose la usabilidad en todo momento cumpliendo con los criterios de la IHC-S.

Existen otros aspectos a considerar como trabajo futuro, como la complementación de la sección de trabajos relacionados con más investigaciones para compararlas con lo presentado en este artículo y con ello mejorar la interfaz propuesta, así mismo, la habilitación de la opción de desactivación del módulo de transacción segura durante toda la transacción para que el usuario o solicitante tenga ese poder de decisión y lo use cuando mas le convenga.

7. Referencias

[1] J. Johnston, J. H. P. Eloff and L. Labuschagne, "Security and human computer interfaces", *Computers & Security Vol. 22, No 8* 0167-4048/03, Elsevier Ltd, 2003, pp. 675-684.

[2] T. T. Hewett, R. Baecker, S. Card, T. Carey, J. Gasen, M. Mantei, G. Perlman, G. Strong and W. Verplank, "ACM SIGCHI Curricula for Human-Computer Interaction", http://www.acm.org/sigchi/cdg/cdg2.html Copyright © 1992, 1996 Last updated: 2004-06-03, (Accesed: 2006-09-27).

[3] B. L. Flores, J. E. Ibarra, J. A. Rodríguez, "Cybermetric's Rol on the Diagnostic of Usability of Web Sites" (In Spanish), *Memorias de las V Jornadas Iberoamericanas de Ingeniería de Software Ingeniería del Conocimiento (JIISIC 06)*, pp. 175-181.

[4] J. Löwgren, "Perspectives on Usability", ISSN-0281-4250 LiTH – IDA - RIDA. *Technical Report* 1995, Department of Computer and Information Science Linköping University.

[5] R. Mendoza, J. Muñoz, F. Álvarez, and M. Vargas, "Exploratory Metrics to Evaluate the Security Factors of Transactions between a Requester and a Web-service", *Technical report poster version*, Latin American Web Congress (LaWeb 2006).

[6] J. Nielsen, "Ten Usability Heuristics", www.useit.com/ papers/heuristic/heuristic_list.html (Accessed: 2006-10-12).

[7] S. D'Hertefelt, "Trust and the Perception of Security", 3 January 2000. http://www.interactionarchitect.com /research /report20000103shd.htm, (Accessed: 2006-10-14).

[8] M. García Ruiz, M. Vargas Martin, M. Green, "Towards a Multimodal Human-Computer Interface to Analyze Intrusion Detection in Computer Networks", MEXIHC Puebla, México. 2006.

Experimento Exploratorio para la Validación de Medidas para Modelos de Procesos de Negocio

Elvira Rolón[(1)], Félix García[(2)], Francisco Ruiz[(2)], Mario Piattini[(2)]
[(1)]Facultad de Ingeniería Arturo Narro Siller
Universidad Autónoma de Tamaulipas
Centro Universitario Tampico-Madero, 89336, México
erolon@proyectos.inf-cr.uclm.es

[(2)]Departamento de Tecnologías y Sistemas de Información
Centro Mixto de Investigación y Desarrollo de Software UCLM-Soluziona
Universidad de Castilla-La Mancha, Paseo de la Universidad 4,
13071, Ciudad Real, España.
{felix.garcia, francisco.ruizg, mario.piattini}@uclm.es

Resumen

En un entorno empresarial de mejora continua, los procesos de negocio requieren de frecuentes cambios en los que se ven afectadas todas las etapas del ciclo de vida del proceso, principalmente la etapa de modelado ó diseño. Esta etapa toma especial importancia al ser la base para que los procesos posteriormente puedan ser entendidos y llevados a cabo.

De cara a promover la mejora de los modelos de procesos de negocio, en este trabajo se presenta un conjunto de medidas para evaluar la complejidad estructural de modelos conceptuales de procesos de negocio. El objetivo es obtener indicadores útiles a la hora de llevar a cabo las labores de mantenimiento de los modelos, así como facilitar la evaluación temprana de ciertas propiedades de calidad del modelo. Para ello, se ha realizado la validación de las medidas propuestas mediante un estudio empírico. Con este estudio fue posible conocer el conjunto de medidas útiles para evaluar la usabilidad y la mantenibilidad de modelos conceptuales de procesos de negocio.

1. Introducción

El modelado de procesos de negocio es uno de los primeros pasos en el logro de las metas de las organizaciones, y sus objetivos, desde el punto de vista empresarial, recaen en diversos aspectos de los cuales destacan dos categorías [1]: a) Mejorar el entendimiento de una situación y comunicarla entre los diversos *stakeholders* y, b) Utilizarlos como una herramienta para alcanzar las metas de un proyecto de proceso de desarrollo. Además, desde el punto de vista de sistemas, se considera que el modelado de procesos de negocio debe ser una parte esencial de cualquier proyecto de desarrollo software, que permita al analista capturar el esquema y los procedimientos generales que rigen al negocio [2].

Adicionalmente a los claros objetivos existentes desde ambos puntos de vista (empresarial y de sistemas), aparecen otros aspectos no menos importantes que los modelos de procesos de negocio deben cubrir. De igual forma que en el área de la ingeniería del software es importante la etapa de mantenimiento de los procesos, nosotros consideramos que esa importancia es equivalente para el caso de los modelos de procesos de negocio.

En la última década, las empresas se han visto envueltas en entornos de competitividad y en constante cambio tanto interno como externo, por lo que con frecuencia es necesario efectuar actualizaciones o modificaciones en sus procesos. Este movimiento de las organizaciones hacia una mejora continua, es lo que se conoce como la iniciativa BPR *(Business Process Re-enginering)*, propuesta por Hammer y Champy en los 90´s [3].

Actualmente, mediante la iniciativa en auge de los últimos años que es la Gestión de Procesos de Negocio *(BPM, Business Process Management),* se abarcan todas las fases del ciclo de vida del proceso, haciendo converger la teoría de la gestión con las nuevas tecnologías [4].

Nuestro interés se centra en la fase de diseño del proceso, mediante la cual se facilita la visualización de las tareas que se llevan a cabo dentro de la organización. Esta fase se refiere al modelado, manipulación y rediseño de los procesos, pero cuando es necesario efectuar tareas de mantenimiento puede llegar a ser una etapa complicada que implica inversión de tiempo y recursos, ya que involucra tanto a desarrolladores técnicos como a analistas de negocios.

Teniendo en cuenta estos factores, nuestro trabajo se enfoca en la evaluación de la complejidad estructural de los modelos de procesos de negocio en un nivel conceptual. Con esto se pretende dar soporte a la gestión de los procesos de negocio facilitando la evaluación temprana de ciertas propiedades de calidad de los modelos. Además, esto facilitaría la evolución de los modelos de procesos de negocio al proporcionar información subjetiva acerca de su mantenibilidad, principalmente en aquellas organizaciones inmersas en una mejora continua.

El punto de partida para este estudio ha sido la definición de medidas para modelos de procesos de negocio expresados con la notación estándar BPMN *(Business Process Modeling Notation)* [5]. Para validar las medidas propuestas, actualmente se esta llevando a cabo una familia de experimentos con una población integrada por expertos en análisis de negocios y en ingeniería del software. Esto nos permitirá comparar los resultados de ambos tipos de perfiles.

El artículo contiene los siguientes apartados: en la sección 2 se hace un resumen de los trabajos relacionados con la evaluación y medición de los procesos de negocio. En la sección 3 se presentan las medidas que hemos definido y a continuación en la sección 4 se describen diversos aspectos del primer experimento que se ha llevado a cabo. En la sección 5 se muestra en análisis de los resultados obtenidos a partir de los datos recopilados en dicho experimento. Finalmente en la sección 6, se plantean las conclusiones y el trabajo por realizar a futuro.

2. Trabajos relacionados

Poco se puede encontrar en la literatura relacionada con la medición y evaluación de los procesos de negocio (PN), al menos en un nivel conceptual como es nuestro tema de estudio. La mayoría de la investigación en este campo se ha centrado en otros aspectos tales como la evaluación de los resultados obtenidos, los tiempos de ejecución, los costos del proceso, etc., es decir el análisis se hace a nivel de la ejecución del proceso.

La medición y control de los PN son temas que han sido abarcados en estudios como el de Powell et al. [6] en el que pretenden identificar mecanismos de control para PN que son efectivos en diferentes tipos de entornos. Por su parte, Tjaden [7], define tres métricas para evaluar la efectividad estructural de los PN a las cuales llamó de complejidad, integración y dinamismo; basándose en la idea de que para ser capaces de predecir la actuación antes de que un nuevo proceso sea implementado la gerencia necesita métricas estructurales que analicen propiedades más estáticas de los procesos de negocio.

Otra propuesta es presentada por Vitolins [8], en donde propone una nueva metodología para definir medidas de PN en base a un metamodelo de medición de procesos de negocio. En [9] se presenta una recopilación de métricas de complejidad de modelos de PN encontradas en la literatura, las cuales fueron comparadas a un conjunto de criterios.

Otras líneas se enfocan a la evaluación de la calidad de las diversas técnicas utilizadas para el modelado de PN, como en [10, 11] donde se propone un marco para su descripción y evaluación, el cual dividen en dos partes: en la manera de modelar y en la manera de trabajar de una técnica de modelado. El objetivo del estudio era el de proporcionar un conjunto de propiedades bien definidas, así como de una serie de procedimientos para hacer una medición objetiva de las mismas.

Un trabajo reciente sobre medidas de complejidad para modelos de PN, es el presentado por Gruhn y Laue [12], en donde discuten cómo las ideas conocidas en la investigación acerca de la complejidad del software, pueden ser usadas para analizar la complejidad de los modelos de procesos de negocio. En base a la idea de que al medir el peso cognitivo del modelo de proceso de negocio (MPN), se puede obtener información acerca de la facilidad o dificultad de comprenderlo, asignando un peso cognitivo a los elementos del MPN, para posteriormente definir el peso cognitivo del modelo en su totalidad.

3. Medidas para modelos de procesos de negocio

Para evaluar la complejidad de los modelos de procesos de negocio, en el trabajo aquí presentado se ha definido un conjunto representativo de medidas.

Las medidas han sido definidas siguiendo la terminología de la notación BPMN 2.0 [5], por ser la notación estándar mas reciente específica para el modelado de procesos de negocio. BPMN proporciona una notación gráfica para expresar procesos de negocio mediante un Diagrama de Procesos de Negocio (DPN) que está compuesto de dos categorías básicas de elementos con los cuales es posible desarrollar desde

modelos de procesos simples hasta modelos de alto nivel.

Nuestro trabajo tiene como punto de partida la propuesta FMESP (Framework for the Modeling and Evaluation of Software Processes) [13], en la cual se definen un conjunto de medidas para evaluar los modelos de procesos software en dos niveles: 1) a nivel de modelo para evaluar la complejidad estructural del modelo en su totalidad; 2) a nivel de los elementos fundamentales del modelo, para evaluar la complejidad concreta de elementos tales como actividades, roles o productos de trabajo. El objetivo de la definición y validación de métricas en FMESP fue el de determinar un conjunto de indicadores que fueran útiles para la mantenibilidad de los modelos de proceso software mediante la evaluación de la complejidad estructural de los mismos.

Con el fin de conocer de forma objetiva cuál es la mantenibilidad de los modelos de procesos de negocio (MPNs) se ha definido un conjunto de medidas para evaluar modelos expresados en BPMN, mediante la adaptación y extensión de las medidas de FMESP a modelos de proceso de negocio. Los términos utilizados en este trabajo en cuanto a la medición de modelos de procesos de negocio, se basa en la Ontología de la Medición del Software definida por García et al. [14]. De este modo, el conjunto de medidas definidas han sido agrupadas en dos categorías: medidas base y medidas derivadas.

Las medidas base consisten principalmente en contar los elementos significativos del modelo de proceso de negocio, de las cuales se ha definido un total 46 medidas base en función de los principales elementos que componen en metamodelo BPMN. Las medidas base están definidas acorde a las cuatro categorías de elementos centrales, quedando distribuidas de la siguiente manera: 37 medidas base corresponden a la categoría de Objetos de Flujo, 5 a la categoría de Objetos de Conexión, 2 a la categoría de Carriles y 2 a la categoría de Artefactos.

A partir de las medidas base se ha definido un conjunto de medidas derivadas, las cuales permiten conocer las proporciones existentes entre los diferentes elementos del modelo. El conjunto de las medidas derivadas que se han definido se muestra en la Tabla 1.

Tabla 1. Medidas derivadas

Medida	Definición y Fórmula
TNSE	Número Total de Eventos de Inicio del Modelo TNSE = NSNE+NSTE+NSMsE +NSRE+NSLE+NSMuE
TNIE	Número Total de Eventos Intermedios del modelo TNIE = NINE+NITE+NIMsE+NIEE +NICaE+NICoE+NIRE+NILE+NIMuE
TNEE	Número Total de Eventos Finales del Modelo TNEE = NENE+NEMsE+NEEE+ NECaE+NECoE+NELE+NEMuE+NETE
TNT	Número Total de Tareas del Modelo TNT = NT+NTL+NTMI+NTC
TNCS	Número Total de Sub-Procesos Colapsados del Modelo TNCS = NCS+NCSL+NCSMI+NCSC+NCSA
TNE	Número Total de Eventos del Modelo TNE = TNSE + TNIE + TNEE
TNG	Número Total de Decisiones/Uniones del Modelo TNG = NEDDB+NEDEB+NID+NCD+NPF
TNDO	Número Total de Objetos de Datos en el Modelo TNDO = NDOIn + NDOOut
CLA	Nivel de Conectividad entre Actividades $CLA = \frac{TNT}{NS}$
CLP	Nivel de Conectividad entre Participantes $CLP = \frac{NMF}{NP}$
PDOPIn	Proporción de Objetos de Datos como productos de entrada y el total de Objetos de Datos $PDOPIn = \frac{NDOIn}{TNDO}$
PDOPOut	Proporción de Objetos de Datos como productos de salida y el total de Objetos de Datos. $PDOPOut = \frac{NDOOut}{TNDO}$
PDOTOut	Proporción de Objetos de Datos Producto resultante de las Actividades del Modelo $PDOTOut = \frac{NDOOut}{TNT}$
PLT	Proporción de Participantes y/o carriles y las actividades del Modelo $PLT = \frac{NL}{TNT}$

El objetivo al definir las medidas base y derivadas descritas, es el de poder evaluar la complejidad estructural de los modelos de proceso de negocio expresados con BPMN. En primera instancia la definición de las medidas nos permitió investigar la relación entre la complejidad estructural y la mantenibilidad como atributo externo de dichos modelos. Al efectuar la validación teórica de las medidas definidas bajo el marco de Briand et al. [15], fue posible agruparlas de acuerdo a diferentes propiedades de la complejidad estructural (Figura 1).

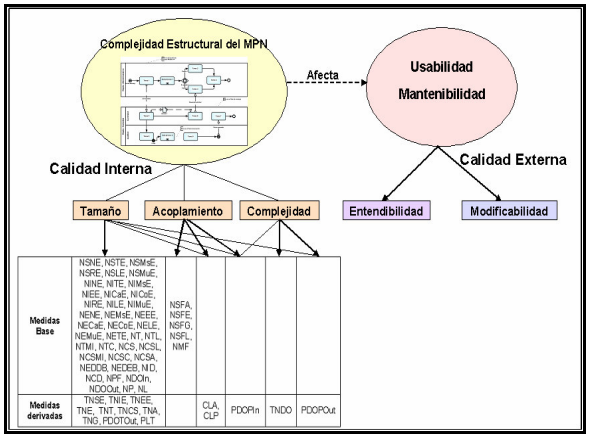


Figura 1. Relación entre la complejidad estructural y los atributos de calidad

4. Primer experimento

En base al trabajo realizado por Canfora et al. [16] y con el objetivo de establecer qué medidas son útiles para evaluar la entendibilidad y mantenibilidad de los MPNs, se ha iniciado el desarrollo de una familia de experimentos, que además nos permitirá evaluar aspectos de calidad de modelos conceptuales de procesos de negocio expresados con BPMN.

Ante la cantidad de medidas propuestas y con el fin de seleccionar un conjunto representativo de ellas, se ha efectuado la validación empírica de las medidas. Inicialmente se realizó un análisis de correlación de los valores de las medidas con respecto a los tiempos de respuesta y al número de aciertos de los resultados obtenidos a partir de un primer experimento, el cual se llevó a cabo siguiendo las sugerencias de Perry et al. [17], Wholin et al. [18], Juristo y Moreno [19], Ciolkowski et al. [20] y Briand et al. [21].

4.1. Objetivos de la investigación

Usando la plantilla GQM (Goal Question Metric) [22], el objetivo del experimento se define como: *Analizar* medidas de complejidad estructural para modelos de proceso de negocio *con el propósito de* evaluarlas en relación a la capacidad de ser usadas como indicadores de la entendibilidad y la modificabilidad de dichos modelos, *desde el punto de vista de* los investigadores *en el contexto de* estudiantes, becarios de investigación y profesores de ingeniería en informática.

4.2. Participantes

El grupo de participantes estuvo formado por 27 sujetos entre estudiantes de doctorado, becarios de investigación y profesores de la Escuela Superior de Informática de la Universidad de Castilla-La Mancha en España.

Los sujetos fueron elegidos por conveniencia y todos contaban con amplios conocimientos sobre el modelado del producto (UML, bases de datos, etc.), pero no tenían conocimientos previos acerca del modelado conceptual de procesos de negocio, por lo que se les impartió una sesión de preparación previa a la realización del experimento en la que se les explicó la notación estándar para el modelado de procesos de negocio BPMN. Sin embargo, a pesar de la sesión de entrenamiento, los sujetos no fueron concientes de los aspectos que intentábamos estudiar.

4.3. Material

El material experimental estuvo compuesto por diez modelos de procesos de negocio expresados con BPMN con diferentes dimensiones y características estructurales entre si, por lo que presentaban distintos grados de complejidad, obtenidos variando el valor de las medidas en cada modelo, tal como se puede apreciar en la Tablas 2 y 3. La intención al elegir modelos de dimensiones distintas, es la de determinar la influencia de la complejidad del modelo para diferentes usuarios como pueden ser los analistas de negocios y los ingenieros de software, en quienes particularmente esta enfocado el objetivo de nuestro estudio.

Tabla 2. Valores de las medidas base

MPN	VALORES MEDIDAS BASE																						
	NSNE	NSMsE	NITE	NIMsE	NICoE	NENE	NEMsE	NECoE	NT	NTL	NCS	NEDDB	NEDEB	NPF	NSFA	NSFE	NSFG	NSFL	NMF	NP	NL	NDOIn	NDOOut
1	2					2			9			1			6	2	2		2	2		1	2
2	2					2			6			1			2	2	2		3	2		2	0
3	2					4			10			1			4	2	6	1	5	2		3	3
4	1					1			5		2	2	2		1	1	8		2	2		0	2
5	3			2		1	2		9		2	3		1	7	3	7		1	2		14	8
6	1	1		1	3	1	1	1	9			3			0	9	6		12	5		1	1
7	1	1	1	1	2	6	1	1	15	2	3	3	1		7	6	11	1	3	3		0	2
8	3					3			20			1			16	3	2	0		3		2	2
9	3					4			27			1			20	3	2	0	12	3	2	2	2
10	3					9			32			3	3		15	3	14	2	16	3		2	3

Tabla 3. Valores de las medidas derivadas

MPN	VALORES MEDIDAS DERIVADAS														
	TNSE	TNIE	TNEE	TNE	TNT	TNCS	TNA	TNG	TNDO	CLA	CLP	PDOPIn	PDOPOut	PDOTOut	PLT
1	2	0	2	4	9	0	9	1	3	1,5000	1,0000	0,3333	0,6667	0,2222	0,0000
2	2	0	2	4	6	0	6	1	2	3,0000	1,5000	1,0000	0,0000	0,0000	0,0000
3	2	0	4	6	10	0	10	1	6	2,5000	2,5000	0,5000	0,5000	0,3000	0,0000
4	1	0	1	2	5	2	7	4	2	7,0000	1,0000	0,0000	1,0000	0,4000	0,0000
5	3	2	3	8	9	2	11	4	22	1,5714	0,5000	0,6364	0,3636	0,8889	0,0000
6	2	4	3	9	9	0	9	3	2	0,0000	2,4000	0,5000	0,5000	0,1111	0,0000
7	2	4	8	14	17	3	20	4	2	2,8571	1,0000	0,0000	1,0000	0,1176	0,0000
8	3	0	3	6	20	0	20	1	4	1,2500	0,0000	0,5000	0,5000	0,1000	0,0000
9	3	0	4	7	27	0	27	1	4	1,3500	4,0000	0,5000	0,5000	0,0741	0,0741
10	3	0	9	12	32	0	32	6	5	2,1333	5,3333	0,4000	0,6000	0,0938	0,0000

Para cada modelo se elaboraron dos cuestionarios: en el primero de ellos se pidió responder a una serie de preguntas relacionas a la entendibilidad del modelo, y en el segundo se propuso una serie de modificaciones a realizar en el modelo. Además, al final de cada cuestionario se incluyó una pregunta, para que los sujetos evaluaran de forma subjetiva la complejidad de los modelos presentados. El material también incluía un ejemplo resuelto en el cual se indicaba la forma en que debían realizarse los ejercicios. Un ejemplo del material experimental se muestra en el Apéndice A.

4.4. Diseño experimental

Se realizó un diseño intra-sujetos, en el que todos los sujetos tenían que contestar a todos los test. Los diez modelos de procesos de negocio que se entregaron a cada sujeto fueron dados en diferente orden. Al repartir el material ya descrito a los sujetos, se hizo una breve explicación de cómo rellenar los test, indicándoles que no había límite de tiempo para la realización de los mismos y que en caso de duda podían preguntar al responsable de la organización del experimento.

Una visión general del diseño del experimento se puede observar en la Figura 2.

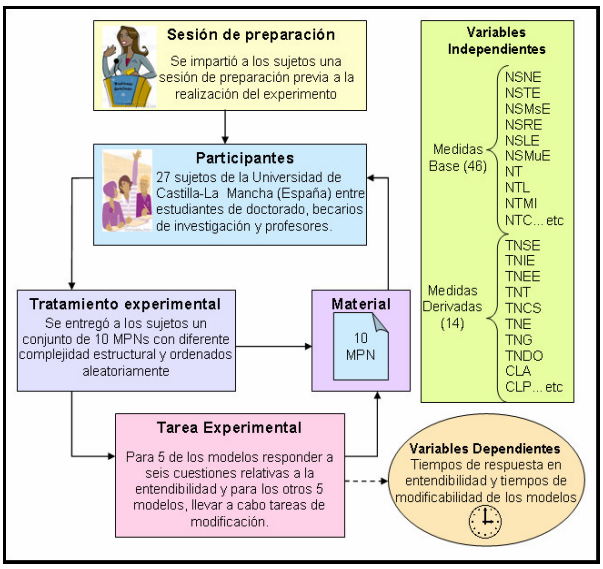


Figura 2. Diseño Experimental

Además de la sesión introductoria en el modelado de procesos de negocio con BPMN, junto con el material experimental, se les hizo entrega a los sujetos de una guía explicativa de la notación BPMN y dos ejercicios resueltos para cada uno de los dos cuestionarios que comprendía el experimento.

4.5. Tarea experimental

Cada sujeto recibió un material compuesto por diez MPNs, cinco de ellos con el cuestionario referente a la entendibilidad del modelo y los otros cinco con los ejercicios de modificabilidad. Para cada cuestionario los sujetos tuvieron que hacer una de las siguientes tareas: responder "si" ó "no" a las seis cuestiones relativas a la entendibilidad del modelo, y llevar a cabo cinco modi-ficaciones consistentes en adicionar y/o eliminar tareas, objetos de datos, roles o dependencias entre elementos.

La tareas a desarrollar en cada aspecto a evaluar (entendibilidad y modificabilidad) fueron similares en cuanto al grado de complejidad, siendo este un aspecto fundamental que se tuvo en cuenta en la preparación del material. Por esta razón, se consideró que la única variación en el esfuerzo para realizar las tareas solicitadas, debía ser la complejidad de cada modelo.

Antes de iniciar las tareas solicitadas en cada cuestionario, se pidió a los sujetos que escribieran la hora de inicio, y al término de la realización de las tareas solicitadas también se les pidió escribieran la hora de finalización. Por último, se les pidió a los sujetos que hicieran una valoración subjetiva de la complejidad total del modelo de acuerdo a su opinión, para lo cual se les presentó una escala compuesta de cinco niveles lingüísticos (desde 1 = Muy simple hasta 5 = Muy complejo). Se seleccionaron cinco etiquetas lingüísticas porque consideramos que eran suficientes para cubrir todas las categorías posibles para cada subcaraterística a evaluar, siguiendo además las recomendaciones de Godo et al. [23] y de Bonissone [24] a la hora de seleccionar un número impar de etiquetas.

4.6. Variables

Las variables independientes se refieren a la complejidad estructural de los MPNs, es decir están relacionadas con las medidas base y las medidas derivadas. Las variables dependientes son las relativas a las dos subcaracterísticas de calidad: la entendibilidad y la modificabilidad de los MPNs, las cuales fueron medidas a través de los tiempos de respuesta empleados por los sujetos para llevar a cabo las tareas requeridas, así como de los aciertos a las cuestiones relacionadas a las tareas de entendimiento y de los aciertos en las tareas de modificación.

4.7. Hipótesis

Las hipótesis planteadas acorde al objetivo de nuestra investigación son las siguientes:

Hipótesis nula, H_{0u}: No hay una correlación significativa entre las medidas de complejidad estructural y el tiempo de entendibilidad.
Hipótesis alternativa, H_{1u}: Hay una correlación significativa entre las medidas de complejidad estructural y el tiempo de entendibilidad.

Hipótesis nula H_{0m}: No hay una correlación significativa entre las medidas de complejidad estructural y el tiempo de modificabilidad.
Hipótesis alternativa, H_{1m}: Hay una correlación significativa entre las medidas de complejidad estructural y el tiempo de modificabilidad.

5. Análisis de resultados del primer experimento

Para la validación de los datos, una vez que éstos fueron recogidos de las hojas de respuestas, se controló que estuvieran completas y se revisaron los aciertos en las respuestas así como los tiempos empleados para la realización de cada ejercicio.

Al efectuar el análisis e interpretación de los datos recogidos, intentamos comprobar las hipótesis formuladas en el apartado 4.7 para lo cual, inicialmente se realizó un resumen con tales datos. Este resumen está compuesto por los valores de las medidas para cada modelo de proceso de negocio (visto en tablas 2 y 3), por las medianas de las puntuaciones dadas por los sujetos a las dos subcaracterísticas analizadas, así como por la media del tiempo de entendibilidad y modificabilidad en cada modelo (Tabla 4).

Tabla 4. Promedios y Desviación Estándar para los tiempos de entendibilidad y modificabilidad

MPN	Tiempo de Entendibilidad		Tiempo de Modificabilidad	
	Media	Des. Est.	Media	Des. Est.
1	121	43	327	172
2	166	42	401	193
3	185	53	291	106
4	149	57	306	127
5	280	80	375	160
6	279	130	345	143
7	221	75	416	102
8	211	83	305	77
9	187	58	392	106
10	238	98	319	107

Como se puede ver en la Tabla 4, los modelos 5, 6 y 10 fueron los mas difíciles de entender por los sujetos, mientras que los modelos 2, 7 y 9 resultaron con mayor complejidad a la hora de llevar a cabo tareas de mantenimiento, en este caso al efectuar las modificaciones solicitadas. Al analizar los valores de las desviaciones estándar, se puede ver que hay una variación, ya que los modelos 6, 8 y 10 presentan una desviación estándar más alta para el caso de la entendibilidad, mientras que los modelos 1, 2 y 5 son los que presentan mayor desviación estándar para las tareas de modificabilidad.

Al analizar estos resultados y considerando también los valores de las medidas presentadas en las Tablas 2 y 3, los modelos 7, 9 y 10 parecen ser los modelos con mas alta complejidad estructural lo cual proporciona alguna evidencia acerca de la influencia de la complejidad estructural de los modelos de procesos de negocio en su mantenibilidad.

En cuanto al resultado de la valoración subjetiva que se pidió a los sujetos que hicieran acerca de la complejidad de los modelos presentados, estos se resumen en la Tabla 5.

Al analizar la mediana de los datos obtenidos en la valoración subjetiva para cada uno de los modelos, se puede ver que los sujetos consideraron que de acuerdo a la escala utilizada, en el caso de la entendibilidad consideran que casi todos los modelos son de complejidad normal, excepto los modelos 2, 3 y 4 que fueron calificados como de complejidad algo simple.

Tabla 5. Valoración subjetiva de la complejidad de los modelos

MPN	Val. Entend.	Val. Modif.
1	3,00	3,00
2	2,00	2,00
3	2,00	3,00
4	2,00	2,00
5	3,00	3,00
6	3,00	3,00
7	3,00	4,00
8	3,00	3,00
9	3,00	3,00
10	3,00	3,50

En el caso de la valoración subjetiva de los modelos en los que debieron hacer tareas de modificación, los modelos 7 y 10 resultaron con la mayor evaluación de complejidad, siendo considerados como "algo complejos". El resto de los modelos fueron valorados de complejidad "algo simple" y "normal"

Al comparar estos resultados con los tiempos relacionados a las tareas de entendibilidad y modificabilidad, existe una coincidencia en los modelos 7 y 10, los cuales aparecen como algunos de los modelos que presentan mayor complejidad.

Adicionalmente, a partir del resumen de los promedios en los tiempos de entendibilidad y de modificabilidad, así como el resumen de los valores de las medidas fue posible realizar un análisis estadístico. Para comprobar si la distribución de los datos obtenidos era normal, se aplicó el test de Kolmogorov-

Smirnov. Como resultado de ello se obtuvo que la distribución era no normal, por lo que se decidió utilizar un test estadístico no paramétrico como el coeficiente de correlación de Spearman con un nivel de significación $\alpha = 0.05$ lo cual indica la probabilidad de rechazar la hipótesis nula cuando es cierta (error de tipo I), es decir, el nivel de confianza es del 95%. Usando el coeficiente de correlación de Spearman cada una de las medidas fue correlacionada separadamente con los tiempos de entendibilidad y modificabilidad. En la Tabla 6 se muestran los resultados del análisis de correlación para los tiempos de entendibilidad, tiempos de modificabilidad, los aciertos en las tareas de entendibilidad y modificabilidad, así como de la valoración subjetiva de los modelos en ambos ejercicios.

Tabla 6. Resultados del análisis de correlación de Spearman

Medida	TEnt	TMod	AEnt	AMod	VEnt	VMod
NIMsE	,742(*) 0,014	0,472 0,168	-0,35 0,321	-0,373 0,288	0,423 0,224	0,343 0,332
NENE	0,049 0,892	0,025 0,946	0,491 0,15	-0,242 0,5	0,193 0,592	,691(*) 0,027
NEMsE	,742(*) 0,014	0,472 0,168	-0,35 0,321	-0,373 0,288	0,423 0,224	0,343 0,332
NT	0,387 0,27	-0,067 0,853	,636(*) 0,048	-0,562 0,091	0,577 0,081	,756(*) 0,011
NEDDB	,744(*) 0,014	0,268 0,454	-0,216 0,549	-0,374 0,287	0,378 0,282	0,485 0,155
NSFA	0,231 0,521	0,073 0,841	,737(*) 0,015	-0,431 0,213	0,572 0,084	0,512 0,13
NSFE	,824(**) 0,003	0,405 0,245	0,252 0,483	-0,484 0,156	,754(*) 0,012	,677(*) 0,031
NSFL	0,225 0,532	-0,135 0,71	-0,023 0,951	-0,384 0,273	0 1	,699(*) 0,025
NDOIn	0,315 0,376	-0,201 0,577	0,06 0,869	-,687(*) 0,028	-0,039 0,914	-0,036 0,922
NDOOut	0,331 0,35	-0,376 0,284	-0,15 0,679	-,708(*) 0,022	0,163 0,653	0,415 0,234
TNSE	0,563 0,09	0,04 0,912	,657(*) 0,039	-,728(*) 0,017	0,63 0,051	0,421 0,226
TNIE	,682(*) 0,03	0,502 0,139	-0,328 0,355	-0,23 0,523	0,423 0,224	0,419 0,228
TNEE	0,556 0,095	0,099 0,786	0,304 0,393	-0,609 0,062	0,426 0,22	,859(**) 0,001
TNE	,835(**) 0,003	0,366 0,298	0,196 0,587	-0,607 0,063	,650(*) 0,042	,862(**) 0,001
TNT	0,387 0,27	-0,067 0,853	,636(*) 0,048	-0,562 0,091	0,577 0,081	,756(*) 0,011
TNA	0,512 0,13	0 1	0,571 0,085	-0,626 0,053	,650(*) 0,042	,783(**) 0,007
TNDO	0,332 0,348	-0,395 0,258	0,069 0,849	-,805(**) 0,005	0,197 0,586	0,273 0,445
CLA	-0,406 0,244	0,103 0,777	-0,366 0,298	0,433 0,211	-,722(*) 0,018	-0,26 0,467
PDOTOut	0,03 0,934	-0,333 0,347	-,768(**) 0,009	-0,152 0,674	-0,114 0,754	-0,034 0,925

Como se puede apreciar en la Tabla 6, existe una correlación (rechazando la hipótesis H_{0u}) entre los tiempos de entendibilidad y las medidas NIMsE, NEMsE, NEDDB, NSFE, TNIE Y TNE. Sin embargo respecto al tiempo empleado por los sujetos en la modificabilidad de los diagramas, el análisis de correlación no arrojó ningún resultado (aceptando la hipótesis H_{0m}), por lo que ninguna medida tiene correlación con dicha variable. Considerando que no existe ninguna correlación de las medidas definidas con respecto a los tiempos de modificabilidad, en futuros experimentos este aspecto será tomado en cuenta a la hora de refinar el material experimental.

En relación a los aciertos en las tareas de entendibilidad y modificabilidad, existe una correlación entre los aciertos de entendibilidad y las medidas NT, NSFA, TNSE, TNT y PDOTOut. Así mismo, existe una correlación entre los aciertos en los ejercicios de modificación de los modelos y las medidas NDOIn, NDOOut, TNSE y TNDO.

Finalmente, en cuanto a las valoraciones subjetivas que los sujetos hicieron de los modelos, existe una correlación entre la entendibilidad y las medidas NSFE, TNE, TNA y CLA, y una correlación entre la modificabilidad y las medidas NENE, NT, NSFE, NSFL, TNEE, TNE, TNT y TNA. De este análisis se puede resumir que las medidas que presentan dos o mas correlaciones con las variables analizadas son las medidas: NT, NSFE, TNSE, TNE, TNT y TNA.

5.1. Amenazas de la validez

Los principales problemas que amenazaron la validez del estudio empírico fueron:

- **Validez Interna.** Como parte del experimento, se controlaron las siguientes variables:
  - *Características de los participantes*. El uso de un diseño intra-sujetos minimizó el posible riesgo de diferencias entre los sujetos.
  - *Complejidad de las tareas*. Las tareas experimentales fueron equivalentes en complejidad para cada grupo de modelos experimentales (entendibilidad y mantenibilidad).
  - *Instrumentación*. Se usaron las mismas técnicas de medición para las variables dependientes e independientes para todos los participantes. El riesgo de error en la medición fue reducido por el cálculo automático de todos los valores.
  - *Capacitación*. A todos los participantes les fue impartida una sesión de preparación previa y recibieron los conocimientos necesarios para llevar a cabo adecuadamente el experimento.
  - *Efectos de aprendizaje*. Los modelos experimentales fueron entregados a los sujetos en un orden aleatorio y solo un tipo de tarea (entendibilidad ó modificabilidad) fue requerida en cada modelo para minimizar los efectos de aprendizaje y secuencia.

- *Control del entorno*. Este hecho no afectó la validez interna ya que el experimento se llevó a cabo bajo condiciones controladas en el que los participantes fueron supervisados por los encargados de experimento.
- *Efectos de fatiga*. El tiempo promedio de la duración del experimento fue de 40 minutos por lo que se evitaron los efectos de fatiga.
- *Error en la medición*. Otra amenaza a la validez interna es el hecho de que los sujetos eran responsables de registrar los tiempos empleados en la realización de las tareas. Esto incrementa el riesgo de error en la medición para la variable dependiente, ya que los sujetos podían quizás haber registrado los tiempos de forma incorrecta. El diseño intra-sujetos ayudó a minimizar esta amenaza porque el posible error de medición podría distribuirse aleatoriamente a través de los niveles de la variable independiente. Además, un reloj digital fue proyectado en la pizarra durante la ejecución del experimento para facilitar a los participantes anotar tiempos exactos.

- **Validez externa.** Se identificaron tres posibles amenazas a la validez externa del estudio empírico:
  - *Los modelos experimentales*. En este primer experimento se utilizaron algunos modelos de procesos de negocio encontrados en la literatura y otros representativos de casos reales, pero para futuros estudios empíricos se deben utilizar modelos de procesos de negocio reales.
  - *Las tareas experimentales*. Los tipos de tareas a realizar en los modelos fueron diseñados para lograr los objetivos de la investigación, pero estas deberían ser adaptadas a situaciones reales en la práctica.
  - *Población muestral*. Una clara amenaza a la generalidad de los resultados de este estudio fue el tipo de sujetos experimentales. La población seleccionada para este experimento fueron estudiantes de doctorado y profesores, lo cual reduce la posibilidad de generalizar los resultados en la práctica. De cualquier forma, esta amenaza se espera reducir en un futuro, al realizar nuevos estudios empíricos con una población formada por gente del ámbito empresarial.

6. Conclusiones y trabajo futuro

Resulta interesante analizar la complejidad estructural de los modelos de procesos de negocio como un punto de partida para su evaluación, así como para llevar a cabo las tareas de mantenimiento. El diseño, evaluación y mantenimiento de los modelos de procesos de negocio abarca diferentes ámbitos, de ahí que este sea un tema que ha generado interés no solo por parte de la gente del mundo de los negocios, sino también por parte de la gente del área de ingeniería del software.

En este trabajo, se ha presentado un conjunto de medidas que han sido definidas en base a la notación estándar BPMN con el objetivo de analizar y evaluar la complejidad estructural de los modelos de proceso de negocio en un nivel conceptual. Así mismo, se pretende analizar atributos de la calidad del modelo tales como la usabilidad y la mantenibilidad, con lo cual se estaría proporcionando el soporte necesario a la hora de llevar a cabo las tareas de mantenimiento de los modelos de proceso de negocio.

Además, se han presentado los resultados de un primer experimento realizado con estudiantes de doctorado, becarios y profesores de la Escuela de Informática de la Universidad de Castilla-La Mancha. Con este primer estudio empírico fue posible saber que, del total de medidas definidas, seis de ellas tienen correlación con los tiempos de entendibilidad, ocho medidas tienen correlación con los aciertos de entendibilidad y modificabilidad, y 9 medidas tienen correlación con la valoración subjetiva acerca de la complejidad de los modelos.

De igual forma fue posible conocer que de las medidas definidas, ninguna de ellas tiene correlación con los tiempos de modificabilidad de los modelos, por lo que para nuevos experimentos será necesario refinar el material experimental de forma que nos pueda proporcionar información relacionada a dicha variable.

En cuanto al trabajo por realizar se tienen contemplados los siguientes aspectos:

- Para poder confirmar los resultados de este primer experimento se realizará una réplica en la Universidad Autónoma de Tamaulipas (México) con estudiantes de la Maestría en Sistemas de Información, que se imparte en la División de Estudios de Posgrado e Investigación de la Facultad de Ingeniería Arturo Narro Siller.
- En la realización de nuevos experimentos, se pretende analizar dos subcaracterísticas más de la calidad del modelo como son la analizabilidad y la facilidad de aprendizaje, las cuales están relacionadas a la usabilidad y a la mantenibilidad del modelo, respectivamente.
- En el contexto de la familia de experimentos se tiene previsto realizar un nuevo diseño experimental con el fin de confirmar si las medidas no validadas en este primer experimento pueden ser útiles para evaluar la usabilidad y mantenibiblidad de los MPNs, o son candidatas a ser descartadas. Para ello se llevará a cabo un nuevo experimento con estudiantes del Master Universitario en Tecnología

del Software, en la Universidad del Sannio (Benevento, Italia).

- Adicionalmente, se está contemplando el desarrollo de los modelos de procesos de negocio de una empresa del sector salud, lo que nos permitirá en estudios futuros, utilizar modelos experimentales de casos reales.

Agradecimientos

Este trabajo ha sido parcialmente financiado por los proyectos ENIGMAS (Junta de Comunidades de Castilla-La Mancha, Consejería de Educación y Ciencia, referencia PBI-05-058), ESFINGE subvencionado por el Ministerio de Educación y Ciencia (Dirección General de Investigación)/Fondos Europeos de Desarrollo Regional (FEDER), referencia TIN2006-15175-C05-05 y COMPETISOFT (Programa Iberoamericano de Ciencia y Tecnología para el Desarrollo, referencia 506PI0287).

Referencias

[1] Multamäki, M., "Objective-driven planning of business process modeling", in Department of Industrial Engineering and Management. Helsinki University of Technology. 2002.

[2] Sparks, G., An Introduction to UML. The Business Process Model. Sparx Systems, www.aparxsystems.com.au

[3] Hammer, M. y Champy, J., "Reengineering the Corporation: A Manifesto for Business Revolution". London: Nicholas Brealey. 1994

[4] Smith, H. y Fingar, P., "Business Process Management: The Third Wave". USA: Meghan-Kiffer Press. ISBN: 0-929652-33-9. 2003

[5] OMG, "Business Process Modeling Notation", Specification. Object Management Group, February, 2006. http://www.bpmn.org/Documents/OMG%20Final%20Adopted%20BPMN%201-0%20Spec%2006-02-01.pdf

[6] Powell, S.G., Schwaninger, M. y Trimble, C., "Measurement and Control of Business Processes". Systems Dynamics Review, 17 (1), pp 63-91. 2001

[7] Tjaden, G.S., "Business Process Structural Analysis", Georgia Tech Center for Enterprise Systems, October, 1999. www.ces.gatech.edu/research.htm

[8] Vitolins, V. "Business Process Measures". In Proceedings of International Conference on BALTIC DB&IS. Riga, Latvia. pp. 186-197, 2004.

[9] Latva-Koivisto, A.M., "Finding a complexity measure for business process models", Research Report. Systems Analysis Laboratory, Helsinki University of Technology, 2001.

[10] Hommes, B.-J. y van Reijswoud, V. "The Quality of Business Process Modelling Techniques: The application of a framework for understanding the quality of UML". International Conference on Information Systems Concepts: An Integrated Discipline Emerging (ISCO). Leiden, The Netherlands: Kluwer. pp. 117-136, 1999.

[11] Hommes, B.-J. y van Reijswoud, V. "Assessing the Quality of Business Process Modelling Techniques". In Proceedings of the 33rd Hawaii International Conference on Systems Sciences (HICSS 2000). Maui, Hawaii, USA: IEEE. pp. 1007-1016, 2000.

[12] Gruhn, V. y Laue, R. "Complexity Metrics for Business Process Models". In Proceedings of 9th International Conference on Business Information Systems (BIS´06). Klagenfurt, Austria. pp. 1-12, 2006.

[13] García, F., Ruiz, F., Piattini, M., Canfora, G. y Visaggio, C.A., "Framework for the Modeling and Evaluation of Software Processes". Journal of Systems Architecture. 2006

[14] Garcia, F., Bertoa, M.F., Calero, C., Vallecillo, A., Ruiz, F., Piattini, M. y Genero, M., "Towards a Consistent Terminology for Software Measurement". Information and Software Technology, 48 (8), pp. 631-644. 2006

[15] Briand, L., Morasca, S. y Basili, V., "Property-Based Software Engineering Measurement". IEEE Transactions on Software Engineering, 22 (1), pp. 68-86. 1996

[16] Canfora, G., García, F., Piattini, M., Ruiz, F. y Visaggio, C.A., "A Family of Experiments to Validate Metrics for Software Process Models". Journal of Systems and Software, 77 (2), pp. 113-129. 2005

[17] Perry, D., Porte, A. y Votta, L., "Empirical Studies of Software Engineering: A Roadmap". Future of Software Engineering, Ed. Anthony Finkelstein. pp. 345-355. 2000

[18] Wohlin, C., Runeson, P., Höst, M., Ohlson, M., Regnell, B. y Wesslén, A., "Experimentation in Software Engineering: An Introduction." Kluwer Academic Publishers. 2000

[19] Juristo, N. y Moreno, A., "Basics of Software Engineering Experimentation": Kluwer Academic Pub. 2001

[20] Ciolkowski, M., Shull, F. y Biffl, S. "A Family of Experiments to Investigate the Influence of Context on the Effect of Inspection Techniques". In Proceedings of the 6th International Conference on Empirical Assessment in Software Engineering (EASE). Keele (UK). pp. 48-60, 2002.

[21] Briand, L., El Emam, K. y Morasca, S., "Theorical and Empirical Validation of Software Product Measures". International Software Engineering Research Network, Technical Report ISERN-95-03. 1995

[22] Basili, V. y Rombach, H., "The TAME Project: Towards Improvement-Oriented Software Environments". IEEE Transactions on Software Engineering, 14 (6), 728-738. 1988

[23] Godo, L., López de Mántaras, R., Sierra, C. y Verdaguer, A., "MILORD: The Architecture and Management of Linguistically Expressed Uncertainly". International Journal of Intelligent Systems, 4, pp 471-501. 1989

[24] Bonissone, P., "A Fuzzy Sets Based Linguistic Approach: Theory and Applications". Approximate Reasoning in Decision Analysis, Gupta & E. Sanchez (Eds), pp. 329-339. 1982

Apéndice A

Cuestionario del grupo X, correspondiente a las tareas de entendibilidad, Modelo de proceso de negocio 1 (Figura 3)

Tareas a realizar:

Anotar la hora de inicio (indique hh:mm:ss): _____

1) Contestar las siguientes preguntas:

1. ¿Se puede realizar la *actividad* **"Expedir comprobante de pago"** si se sigue la opción de "pago en efectivo" en el nodo posterior a la *actividad* **"Cobrar tasas correspondientes"**?
2. ¿El *objeto de datos* **Comprobante de pago** es producto de salida de la *actividad* **"Cobrar tasas correspondientes"**?
3. ¿El *evento de inicio* del proceso del Solicitante es disparador de la *actividad* **"Revisar documentos"**?
4. ¿Cuando se ejecuta la *actividad* **"Expedir comprobante de pago"** ya se ha realizado la *actividad* **"Revisar Documentos"**?
5. ¿La *actividad* **"Recibir licencia de conducir"** realizada por la *entidad* **Solicitante** es comunicada (enviada) a la *entidad* **Oficina de tráfico**?
6. ¿Puede la *entidad* **Oficina de Tráfico** realizar la *actividad* **"Recibir licencia de conducir"**?

Anotar la hora de finalización (indique hh:mm:ss): _____

2) Según su criterio valore la COMPLEJIDAD del Modelo de Proceso de Negocio.

Muy Simple ☐ **Algo Simple** ☐ **Normal** ☐ **Algo Complejo** ☐ **Muy Complejo** ☐

Cuestionario del grupo Y, correspondiente a las tareas de modificabilidad, Modelo de proceso de negocio 1 (Figura 3).

Tareas a realizar:

Anotar la hora de inicio (indique hh:mm:ss): _____

1) Realizar las modificaciones necesarias para satisfacer los siguientes requisitos

1. Se desea incluir una nueva *actividad* **"Requerir licencia vencida"** posterior a la *actividad* **"Hacer fotografía y recabar firma"**, precedente a **"Entregar licencia de conducir"** y que reciba como entrada el *objeto de datos* **"Licencia vencida".**
2. Se debe indicar con el *objeto de datos* **"Nueva Licencia"** que es un producto de salida de la *actividad* **"Entregar licencia de conducir".**
3. Se desea agregar una nueva *entidad* denominada **"Tesorería"** que recibe de la *entidad* **"Oficina de Tráfico"** información proveniente de la *actividad* **"Expedir comprobante de pago"**.
4. Se desea incluir un *evento intermedio de tiempo* a la *actividad* **"Hacer fotografía y recabar firma"** que indique que dicha actividad demora 15 minutos.
5. Se desea incluir una nueva *actividad* **"Pagar tasas"** posterior a la *actividad* **"Entregar documentos"** y precedente a **"Recibir licencia de conducir"**

Anotar la hora de finalización (indique hh:mm:ss): _____

2) Según su criterio valore la COMPLEJIDAD del Modelo de Proceso de Negocio.

Muy Simple ☐ **Algo Simple** ☐ **Normal** ☐ **Algo Complejo** ☐ **Muy Complejo** ☐

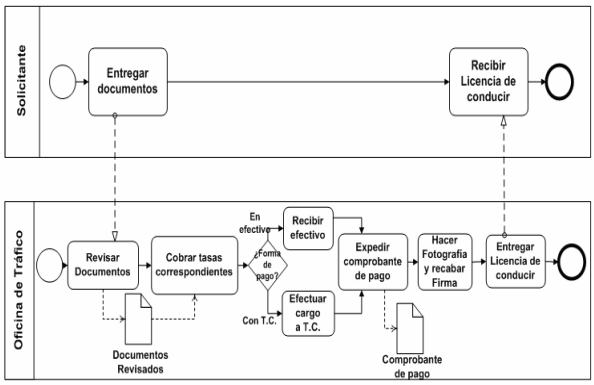


Figura 3. Material experimental: Modelo de proceso de negocio 1

Estudio Experimental en Equipos de Desarrollo de Software sobre las Relaciones entre Personalidad, Satisfacción y Calidad del Producto

Marta Gómez
Escuela Politécnica Superior
Universidad San Pablo-CEU
mgomez.eps@ceu.es

Silvia T. Acuña
Escuela Politécnica Superior
Universidad Autónoma de Madrid
silvia.acunna@uam.es

Ramón Rico
Facultad de Psicología
Universidad Autónoma de Madrid
ramon.rico@uam.es

Resumen

Este artículo presenta un estudio experimental para analizar la personalidad de los integrantes de los equipos que desarrollan software, las características de la tarea y los factores grupales en relación con tanto la calidad de los productos software resultantes como la satisfacción de los integrantes del equipo. Los datos fueron obtenidos de una muestra de 35 equipos de estudiantes de una universidad española.

Los equipos más satisfechos con su trabajo son aquellos que consiguen mayores puntuaciones en los factores de personalidad de amabilidad y el sentido de la responsabilidad. La satisfacción también aumenta cuando los integrantes deciden sobre la forma en que desarrollan y organizan su trabajo. Por el contrario, el grado de satisfacción y cohesión disminuye cuanto mayor es el conflicto de tarea entre los miembros del equipo. Finalmente, los equipos muestran una correlación positiva entre el factor de personalidad extroversión y la calidad del producto software desarrollado.

1. Introducción

Las personas son un factor crítico para el éxito o fracaso del desarrollo de software. Hay investigaciones que incorporan las personas en el proceso software [1][2][3]. Éstas analizan a las personas individualmente y establecen relaciones con las actividades que se realizan dentro del proyecto de software. Hay acuerdo sobre el hecho de que las personas trabajan juntas para realizar tareas de desarrollo interdependientes y estas relaciones son complejas. Esto conduce a la necesidad de examinar la formación de equipos de desarrollo de software. La importancia del estudio de formación de equipos de desarrollo de software radica en que si llegamos a conocer los factores que influyen sobre una mayor eficacia de los equipos de desarrollo, los gestores de software podrán usar este conocimiento para formar mejores equipos.

La psicología social actualmente investiga la formación de equipos considerando un conjunto de factores que afectan el funcionamiento del equipo. Estos trabajos analizan los factores de personalidad de los miembros del equipo y su relación con las características de la tarea [4][5][6]. Sin embargo, en esta relación intervienen otros factores del comportamiento del equipo tales como las interacciones entre las personas, el conflicto, la cohesión, la cooperación y la comunicación [7].

Los resultados de los estudios de psicología social dependen de la tarea que se realiza (aunque existen tendencias generales que parecen afectar por igual a todas las tareas). Por tanto, se hace necesario estudiar en particular equipos que desarrollan software, con el fin de descubrir los factores que afectan en este caso particular y cuánto se parecen estos resultados particulares a los de otros estudios.

En el ámbito del desarrollo del software hay pocos estudios sobre la influencia que tienen algunos factores grupales: cohesión, conflicto, estructura del grupo, coordinación, experiencia, etc., sobre la eficacia y la eficiencia del equipo [8][9]. Otras investigaciones analizan las cuestiones de motivación que gobiernan el comportamiento de los desarrolladores y sus equipos, e incluyen a otras personas de la organización (líderes de equipo, clientes, usuarios, etc.) y cómo su comportamiento puede afectar al desarrollo del software [10].

Este artículo plantea como cuestión principal que la Calidad del producto software obtenido y el nivel de Satisfacción de los integrantes del equipo dependen de los factores de personalidad de los miembros del equipo. Para ello se presenta un estudio experimental que analiza la personalidad de los integrantes de los equipos que desarrollan software, las características de

la tarea (interdependencia y autonomía) y los factores grupales (cohesión y conflicto) en relación con tanto la Calidad de los productos software resultantes como la Satisfacción de los integrantes del equipo. El estudio mide las personalidades de los integrantes de cada equipo en base a los cinco factores de personalidad o Big Five [11]: Extroversión, Neuroticismo, Amabilidad, Sentido de la Responsabilidad y Apertura a la Experiencia. También se miden las características de la tarea que realizan los equipos, considerando sólo la Interdependencia y la Autonomía, como las más relevantes sobre el comportamiento del equipo, tal y como establecieron Molleman et al. [4] en sus investigaciones. La Autonomía de la tarea se refiere a la percepción sobre cuánta libertad tiene un equipo para tomar decisiones sobre objetivos ("qué"), métodos de trabajo ("cómo"), planes de entrega ("cuándo") y distribución de trabajo entre los miembros del equipo ("quién") [12]. La Interdependencia de la tarea se refiere a la situación en la cual el proceso y el resultado de una tarea afecta al proceso y resultado de otras tareas. Esta interdependencia es recíproca lo que significa que las personas son mutuamente interdependientes. Por último, se incluyen otros factores importantes para comprender mejor estas relaciones como son los factores de comportamiento de grupo: Conflicto y Cohesión. El Conflicto aparece cuando surge algún tipo o nivel de desacuerdo o animosidad entre un número de individuos que trabajan juntos en un grupo hacia una meta común. La Cohesión es una medida de integridad del grupo y de su fortaleza, es decir, es el resultado del esfuerzo de los integrantes del grupo por permanecer en él.

Los resultados del estudio apuntan que la Extroversión media en los equipos ayuda a predecir la Calidad del producto software desarrollado. Mientras que la Amabilidad y el Sentido de la Responsabilidad son pronosticadoras de la Satisfacción del trabajo en equipo. Además, la Interdependencia y la Autonomía se relacionan positivamente con el grado de Satisfacción del equipo. Por último, el Conflicto en la realización de este tipo de desarrollo disminuye la Cohesión del equipo y, por tanto, la Satisfacción del equipo de desarrollo de software.

Este artículo está estructurado como sigue. La Sección 2 describe los trabajos relacionados con la formación de equipos en el campo de la ingeniería de software. La Sección 3 detalla el método del estudio experimental llevado a cabo para relacionar los factores de la personalidad con la calidad del software y la satisfacción de los equipos, así como también las características de la tarea y algunos procesos grupales. La Sección 4 presenta el análisis de datos y los resultados. En la Sección 5 se discuten los resultados obtenidos y se dan las conclusiones.

2. Trabajos Relacionados con la Formación de Equipos

Esta sección analiza estudios relacionados con la formación y comportamiento de equipos, en el ámbito de la ingeniería de software, destacando únicamente los factores más relacionados con nuestro estudio, es decir, los cinco factores de Personalidad, la Cohesión, el Conflicto y las características de la Tarea (Autonomía e Interdependencia).

Citando a DeMarco y Lister [13], "La mayoría de los proyectos de desarrollo de software fracasan debido a inconvenientes con los equipos que los realizan". En la actualidad hay un extendido reconocimiento de que la productividad y la eficacia del proceso software son críticos y dependientes de los factores humanos y sociales [14]. La mayor parte de las investigaciones examinan las cualidades individuales de las personas implicadas en el proceso software, considerando los factores de personalidad y las capacidades requeridas según las características de la tarea que se realiza [2][3]. Tenemos que dar un paso más y examinar el equipo de desarrollo, sus interrelaciones y características de personalidad para conocer mejor qué factores influyen en el funcionamiento del mismo.

Existe poca investigación sobre aspectos de grupo aplicados al desarrollo de software. Algunos estudios usan una prueba estándar tal como el MBTI (Myers-Briggs Type Indicador) [15] [16] [17], para determinar las directrices para el éxito del equipo según los tipos de personalidad de los ingenieros del software. Otro estudio determina la relación entre capacidades, rasgos de personalidad y el funcionamiento de equipo [18].

También hay métodos de formación de equipos basados en un modelo de capacidades cuantitativo [19], pero no consideran aspectos como factores de personalidad de los miembros del equipo. Otro método de formación de equipos está basado en el análisis de habilidades requeridas y disponibles [20], aunque no indica cómo evaluar las habilidades de las personas, y tampoco considera las características del grupo y la tarea.

Zuser y Grechening [21] proponen el empleo de un cuestionario basado en las capacidades y los rasgos de personalidad que durante el desarrollo proveen al equipo de información sobre los proyectos de software terminados para mejorar el funcionamiento del equipo. El equipo se construye según las fases de formación de equipo del modelo de Tuckman: formación,

conocimiento mutuo, establecimiento de normas, ejecución y continuidad [22].

El estudio realizado por Yang y Tang [8] examinaba la estructura del equipo y su rendimiento para el desarrollo de un sistema informático desde el punto de vista social, considerando las interacciones (cohesión y conflicto) entre todos los miembros del equipo de desarrollo, desde los usuarios hasta los jefes de proyecto, pasando por el resto de roles: programadores, analistas, jefes de proyecto, etc. El principal objetivo del estudio fue analizar las relaciones entre los aspectos grupales y la estructura del equipo con respecto a su rendimiento. Las principales conclusiones obtenidas fueron:

1. La cohesión del equipo se relacionaba positivamente con el rendimiento.
2. La estructura del equipo fue un factor importante para el rendimiento del equipo.

A pesar de su importancia, hay poca investigación sobre el ajuste persona-equipo en el desarrollo de software. Esperamos que este estudio experimental proporcione una herramienta para su uso en situaciones de formación de equipos en el desarrollo de software.

3. Planteamiento y Diseño del Estudio Experimental

El estudio experimental realizado responde a un cuasiexperimento [23]. Los cuasiexperimentos se realizan cuando no se pueden asignar los sujetos a una condición experimental, es decir, asignar un tratamiento a un grupo. Un cuasiexperimento se caracteriza por ser no intrusivo y poco costoso. Sobre los datos recolectados se aplican las pruebas estadísticas seleccionadas.

A continuación, se describen los participantes, las variables respuesta y los instrumentos de medida utilizados en el cuasiexperimento.

3.1. Participantes y Caracterización del Desarrollo

Los participantes fueron estudiantes de 2º curso de Ingeniería Informática de la Escuela Politécnica Superior de Informática de la Universidad Autónoma de Madrid, durante el curso 2004/2005. Estos estudiantes estaban cursando la asignatura de Estructura de Datos y Algoritmos. El proyecto era realizar el diseño y la implementación de un sistema de software de complejidad media siguiendo una aproximación de la metodología ágil Extreme Programming (XP) [24]. La adaptación de XP realizada permite que el desarrollo del software sea un trabajo en el que colaboran todos los miembros del equipo, formado, en este caso, por tres personas. Cada equipo establece su plan de trabajo, unas veces todo el equipo trabaja simultáneamente sobre el mismo diseño, algoritmo, código o prueba, mientras que en otras ocasiones existe un reparto de tareas y los desarrollos realizados deben de ser comprobados por los demás compañeros antes de su integración. Los roles de "coach" o "consejero" y de "cliente" son desempeñados por profesores de la asignatura. Por tanto, son ajenos a los equipos pero comunes a todos ellos. Según esto, las prácticas de XP llevadas a cabo fueron: propiedad colectiva del código, integración continua, reglas justas y estándares de codificación. Todos los participantes recibieron la misma formación sobre la metodología XP.

El proyecto tuvo una duración de 4 meses y los productos resultantes tuvieron un tamaño entre 87,15 y 130 puntos de función ajustados.

El número de sujetos participantes en este estudio experimental fue de 105, de los cuales 83 fueron varones (79%) y 22 mujeres (21%). El 75% (79 estudiantes) tenían menos de 21 años y el 25% (26 estudiantes) entre 21 y 30 años de edad.

El grupo de estudiantes participantes fue dividido en 35 equipos (de tres integrantes) para trabajar de esta forma en el desarrollo del proyecto de software. Estos equipos se formaron aleatoriamente y sus integrantes estaban ciegos a las condiciones e hipótesis del cuasiexperimento realizado.

3.2. Variables Respuesta: Calidad del Producto Software y Satisfacción

En este estudio, una de las variables respuesta es la Calidad del producto software evaluada a través del análisis del código, la documentación del proyecto y la participación observada de los miembros de los equipos. La calificación de los proyectos desarrollados por los equipos se determinó según la siguiente fórmula ponderada:

Calificación = ((((Modularización * 2 + Pruebas * 2 + Funcionalidad * 2 + Reutilización * 2 + Estilo * 2) / 4) * 0,8) + ((Participación * 10 / 4) * 0,2))

La otra variable respuesta es la Satisfacción de los miembros de los equipos de desarrollo. Se utilizó como instrumento de medida el Cuestionario de Gladstein para evaluar la Satisfacción del equipo [25]. En este cuestionario se indica el grado en que las personas se muestran satisfechas con sus compañeros de equipo, el trabajo conjunto, etc. La escala va de 1

(completamente en desacuerdo) a 5 (completamente de acuerdo).

3.3. Instrumentos de Medida

Para estudiar la personalidad de los participantes de cada equipo se utilizó el Test de Personalidad NEO FFI [11]. Este test, que se realizó antes de comenzar a trabajar en equipo, se compone de 60 preguntas que miden los cinco factores de personalidad a través de 12 ítems cada uno: Neuroticismo (N), Extroversión (E), Apertura a la Experiencia (O), Amabilidad (A) y Sentido de la Responsabilidad (C). Todos los factores son evaluados mediante escalas tipo Likert; al estudiante se le indica que no hay contestaciones correctas o incorrectas y que tiene que prestar mucha atención para contestar con precisión y sinceridad, teniendo en cuenta la puntuación de la escala (1 en total desacuerdo, 5 totalmente de acuerdo).

Los procesos grupales correspondientes a la Cohesión y Conflicto se midieron a través de 13 ítems del Cuestionario de Cohesión Gross [26] y del Cuestionario de Conflicto Intragrupal de la Facultad de Psicología de la Universidad Autónoma de Madrid. La Interdependencia de la Tarea se determinó siguiendo el Cuestionario de Van der Vegt et al. [27], en una escala que va de 1 (completamente en desacuerdo) a 5 (completamente de acuerdo). Por último, la Autonomía de la Tarea se midió a través del Cuestionario de Molleman [12].

3.4. Validez del Cuasiexperimento

En los cuasiexperimentos, al igual que en el caso de los experimentos, hay que evaluar tanto la validez interna como la validez externa. Sin embargo, por una parte, en los diseños cuasiexperimetales existen dificultades para alcanzar las condiciones necesarias para el establecimiento de una relación causal entre las variables independiente y dependiente (validez interna). Por otro lado, ofrecen menor dificultad, en comparación con los experimentos, para que se puedan generalizar los resultados de la investigación (validez externa).

Con respecto a la validez interna, las amenazas consideradas antes de realizar el cuasiexperimento fueron:

1. No es posible garantizar que los estudiantes realicen todas las tareas especificadas en el orden requerido ni que se distribuyan el trabajo adecuadamente. Esto se mitigó asignando dos profesores por cada 60 estudiantes que se encargaron de realizar el seguimiento del desarrollo del proyecto.
2. El nivel de conocimientos sobre la tarea a desarrollar no es el mismo para todos los equipos. Esto se mitigó formando los equipos aleatoriamente y organizando sesiones de formación sobre la metodología XP aplicada en el desarrollo.
3. El proyecto a desarrollar, es decir, se trata de ver las funcionalidades que deben realizar los participantes durante el proyecto. Si los equipos trabajan sobre distintos problemas, los resultados de dichos equipos podrían ser no comparables. Para evitar esta amenaza todos los equipos abordaron el mismo problema o proyecto.

Con respecto a la validez externa, todos los resultados son generalizables en el ámbito académico. La generalización en el ámbito empresarial requiere del planteamiento y diseño de estudios cuasiexperimentales concretos en organizaciones desarrolladoras de software.

4. Análisis de Datos y Resultados

Para el análisis estadístico de los datos y la obtención de resultados se han aplicado las siguientes técnicas estadísticas:

1. Análisis de correlaciones de los factores de personalidad, Autonomía e Interdependencia de la tarea, procesos grupales (Cohesión y Conflicto), Satisfacción y la Calidad del producto software obtenido.
2. Regresión entre la Satisfacción, Conflicto Personal, Conflicto de Tarea, Cohesión e Interdependencia de la tarea.

Las hipótesis de este estudio se definen como:

HA1: Existe relación entre los factores de Personalidad y la Calidad del software desarrollado.
HA2: Existe relación entre los factores de Personalidad y la Satisfacción en los equipos de desarrollo de software.
HA3: Existe relación entre las características de las tareas (Autonomía e Interdependencia) y la Satisfacción en los equipos de desarrollo de software.
HA4: Existe relación entre los procesos grupales (Conflicto y Cohesión) dentro del equipo y la Satisfacción del equipo de desarrollo de software.

A continuación se detallan los resultados obtenidos para cada análisis.

4.1. Correlaciones de los Factores de Equipo

El análisis de correlaciones entre los factores de Personalidad, Autonomía e Interdependencia de la tarea, procesos grupales (Cohesión y Conflicto) y

Satisfacción, reveló algunas asociaciones positivas y significativas (ver Tabla 1).

En la Tabla 1, se observa que en los grupos hay una correlación positiva entre Extroversión con otros dos factores de personalidad, Apertura a la Experiencia (r=0,414 Sig. 0,014) y Amabilidad (r=0,480 Sig. 0,003) y además con la Cohesión del equipo (r=0,469 Sig. 0,005). Del mismo modo, el Neuroticismo presenta correlaciones negativas con el Sentido de la Responsabilidad (r=-0,636 Sig. 0,000) y la Extroversión (r=-0,336 Sig. 0,049).

El Conflicto evaluado presenta una correlación significativa negativa con la Cohesión (r=-0,496 Sig. 0,002) y la Satisfacción (r=-0,464 Sig. 0,005). Dentro del conflicto diferenciamos, el Conflicto Personal y el Conflicto de Tarea, que posteriormente analizaremos.
Los estudiantes muestran una correlación positiva entre su Satisfacción media y la media del equipo en Amabilidad y Sentido de la Responsabilidad, (r=0,334 Sig. 0,050 y r=0,341 Sig. 0,045). La Satisfacción correlaciona también positivamente con la Interdependencia (r=0,797 Sig. 0,000) y la Autonomía (r=0,471 Sig. 0,004) de la tarea.

	N	E	O	A	C	AUTONOMÍA	INTERDEPENDENCIA	SATISFACCIÓN	CONFLICTO	COHESIÓN
N	1	**-0,336***	-0,112	-0,151	**-0,636****	-0,148	-0,214	-0,129	-0,034	-0,186
E		1	**0,414***	**0,480****	0,256	0,302	0,281	0,153	-0,192	**0,469****
O			1	0,302	0,052	-0,085	0,090	0,068	0,236	-0,061
A				1	0,167	**0,503****	**0,500****	**0,334***	-0,157	**0,376***
C					1	0,216	**0,476****	**0,341***	0,007	0,261
AUTONOMÍA						1	**0,599****	**0,471****	-0,314	0,333
INTERDEPENDENCIA							1	**0,797****	-0,243	**0,421***
SATISFACCIÓN								1	**-0,464****	0,294
CONFLICTO									1	**-0,496****
COHESIÓN										1

sig. 0,05*
sig. 0,01**

Tabla 1. Matriz de correlaciones entre los factores de Personalidad, Características de la Tarea, Procesos Grupales y Satisfacción

La Interdependencia de los grupos está relacionada con la Autonomía (r=0,599 Sig. 0,000), la Amabilidad (r=0,500 Sig. 0,002), el Sentido de la Responsabilidad (r=0,476 Sig. 0,004) y la Cohesión (r=0,421 Sig. 0,012). No se han encontrado relaciones significativas entre la Calidad del software obtenido y otros factores evaluados como Autonomía, Interdependencia, Satisfacción, Conflicto o Cohesión. Sin embargo, el conjunto de los equipos evaluados muestran una correlación significativa positiva entre el factor de personalidad Extroversión y la Calidad del Software desarrollado (r=0,455 Sig. 0,038), como se muestra en la Tabla 2. Por último, analizando el rendimiento individual de los 105 estudiantes con respecto a su rendimiento grupal se observa una relación significativa positiva (r=0,201 Sig. 0,040) en la Tabla 3.

	CALIDAD DEL SOFTWARE
EXTROVERSIÓN	0,455*

sig. 0,05*

Tabla 2. Correlación entre Extroversión y Calidad del Software

	RENDIMIENTO GRUPAL
RENDIMIENTO INDIVIDUAL	0,201*

sig. 0,05*

Tabla 3. Correlación entre el Rendimiento Individual y Grupal

4.2. Regresión Lineal de Satisfacción

En la regresión se procedió con un modelo hacia atrás donde se introdujeron todas las variables independientes y después se elimina una a una basándose en los criterios de salida (R2 más bajo en valor absoluto; criterio de F >= 0,1). De esta forma se obtiene el modelo de la Tabla 4.

	Coeficientes de regresión sin estandarizar	Coeficientes de regresión estandarizados
	B	**Beta**
(Constantes)	8,013	
INTERDEPENDENCIA	0,521	0,765
COHESIÓN	-0,168	-0,182
CONFLICTO DE TAREA	-0,674	-0,380

R^2 = 0,733

Tabla 4. Modelo de Regresión para Satisfacción

Este procedimiento se realizó en dos pasos, llegando a un coeficiente de determinación o correlación R2 corregida de 0,733. Es decir, el modelo predice o explica en ese grado la varianza de la

variable Satisfacción (VD) a partir de la Interdependencia, Cohesión y Conflicto de Tarea, eliminándose el Conflicto Personal. Se resalta como el Conflicto Personal evaluado no contribuye a la explicación del modelo.

Por lo tanto, la variable con más peso predictivo en la Satisfacción evaluada es la Interdependencia con un coeficiente Beta = 0,765. Justamente, al ser equipos pequeños, que realizan actividades que deben concluir en un período de tiempo muy corto y con éxito muy rápido (producto de calidad a muy corto plazo), la Interdependencia es imprescindible pues el grado de frustración de no existir la misma sería muy grande al no conseguir el objetivo.

La Tabla 5 muestra como la regresión lineal converge con las correlaciones entre las variables del modelo, donde la Interdependencia y Satisfacción correlacionan (r=0,797 Sig. 0,000). Vemos también como el Conflicto de Tarea (r=-0,526 Sig. 0,001) está más relacionado negativamente que el Conflicto de Personal (r=-0,354 Sig. 0,037). Es decir, según la Tabla 5, si existe conflicto entre las personas integrantes del equipo, parece lógico que sea más difícil conseguir acuerdos para resolver la tarea a realizar.

	SATISFACCIÓN	CONFLICTO PERSONAL	CONFLICTO TAREA	INTERDEPENDENCIA
SATISFACCIÓN	1	-0,354*	-0,526**	0,797**
CONFLICTO PERSONAL		1	0,637**	-0,178
CONFLICTO TAREA			1	-0,287
INTERDEPENDENCIA				1

sig. 0,05*

sig. 0,01**

Tabla 5. Correlaciones entre Satisfacción y Conflicto

5. Discusión y Conclusiones

La Figura 1 muestra gráficamente los resultados obtenidos en nuestro estudio. Podemos destacar que los factores de personalidad que muestran mayor número de correlaciones son la Amabilidad y la Extroversión. En la Figura 1 también destaca la importancia del factor grupal Cohesión, tanto por sus relaciones con los factores de personalidad anteriormente indicados como con respecto a las características de la tarea.

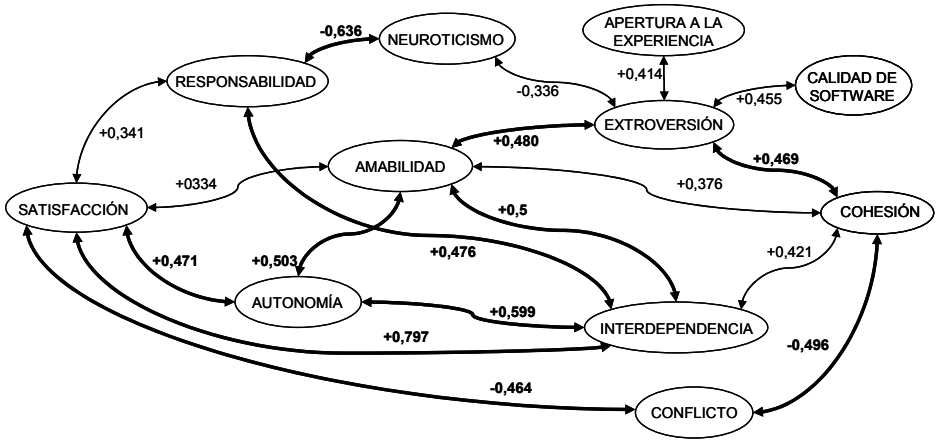


Figura 1. Correlaciones de Personalidad-Comportamiento Grupal-Características de la Tarea-Calidad/Satisfacción

En la Figura 1, sobre los distintos factores de personalidad se observan varias relaciones, algunas más fáciles de comprender e interpretar que otras. Así por ejemplo, el factor de personalidad Extroversión aparece claramente relacionado con Apertura a la Experiencia y Amabilidad. Esto se puede entender si pensamos que en equipos con niveles de extroversión media las relaciones tienen lugar de una forma más amigable y fácil, sin tensiones entre los miembros. Por otro lado, parece razonable que con este ambiente de equipo sea más fácil que las personas propongan incorporar innovación sobre la forma de desarrollar su trabajo. Por tanto, es posible que en este tipo de equipos se produzca la mejora de la calidad del trabajo y como consecuencia la mejora de su resultado, en este caso el producto software desarrollado. Además, es normal que con niveles de Extroversión media unido a los factores de Apertura a la Experiencia y Amabilidad entre sus miembros, la Cohesión del equipo se fortalezca.

En relación con la Satisfacción podemos señalar varias cosas (Figura 1). Se aprecia que los estudiantes más satisfechos con su trabajo son justamente aquellos que consiguen mayores puntuaciones en los aspectos de Amabilidad y Sentido de la Responsabilidad. Esta relación positiva entre Satisfacción y Sentido de la

Responsabilidad ha sido comprobada también por Molleman et al. [4]. Es decir, en los equipos donde se da un ambiente más amigable y/o más responsable, sus miembros se sienten más satisfechos. Los niveles de Satisfacción son también más altos cuando el equipo percibe que puede decidir sobre la forma de desarrollar y organizar el trabajo a realizar (Autonomía). Finalmente, cuanto mayor se percibe el Conflicto en el equipo, la Cohesión entre sus miembros y el grado de Satisfacción que sienten por su trabajo disminuyen.

En relación con la Calidad del software sólo se encontró una clara relación con la Extroversión para los equipos considerados (Figura 1). Las puntuaciones obtenidas en algunos factores del NEO-FFI no se encuentran entre los valores medios estandarizados para la muestra considerada en este estudio. Una muestra tan homogénea no facilita el contraste de las hipótesis del estudio. Cabe pensar que ante una mayor variabilidad de la muestra, los resultados podrían haber sido más claros. Esto explicaría, al menos en parte, la falta de relaciones estadísticamente significativas con la Calidad del software desarrollado, tal y como ocurre en otras investigaciones. No obstante, la relación comprobada coincide con los resultados obtenidos por Barrick & Mount [6] Barry & Stewart [5] y por Barrick et al. [7]. Aunque estos trabajos tratan distintos tipos de tareas, la característica común es la alta interacción entre los miembros del equipo y una visión unánime de los objetivos a alcanzar. Estas son características esenciales también para el modo de desarrollo ágil. Por tanto, se considera que equipos con extroversión media para este modo de desarrollo, llevando a cabo proyectos de corta duración, favorece la calidad del producto software obtenido.

Por último, se puede señalar que se ha considerado tanto la validez interna como la validez externa del cuasiexperimento. Así, para la validez interna se han considerado los riesgos previstos antes de realizar el cuasiexperimento y la forma de evitar su influencia sobre los resultados del mismo. En primer lugar se pensó en la forma de trabajar de los equipos, cómo realizaban las tareas especificadas, en qué orden y la forma de distribuirse el trabajo. Para resolver sus posibles efectos dos profesores hicieron el seguimiento del desarrollo del proyecto. En segundo lugar se consideró el efecto que podría tener la experiencia y conocimiento de los integrantes de los equipos en el proyecto a desarrollar y en su calidad. La forma de neutralizar esto fue realizando sesiones de formación comunes a todos los participantes y luego distribuir aleatoriamente a los miembros en los diferentes equipos. Finalmente, se tuvo en cuenta el proyecto a desarrollar, estableciéndose un mismo proyecto para todos los equipos a fin de que los resultados obtenidos pudieran ser comparables. En relación con la validez externa, las conclusiones obtenidas son generalizables en el ámbito académico para el desarrollo de proyectos software de tamaño pequeño-medio y siguiendo la adaptación de la metodología ágil XP realizada.

Referencias

[1] M.I. Kellner, R.J. Madachy, D.M. Raffo, "Software Process Simulation Modelling: Why? What? How?", *Journal of Systems and Software,*1999, Vol. 46, pp. 91-105.

[2] J. Wynekoop, D. Walz, "Investigating traits of top performing software developers", *Information Technology & People*, 2000, Vol. 13(3), pp. 186-195.

[3] S.T. Acuña, N. Juristo, "Assigning people to roles in software projects", *Software: Practice and Experience*, 2004, Vol. 34, pp. 675-696.

[4] E. Molleman, A. Nauta, K.A. Jehn, "Person-Job Fit Applied to teamwork: A Multi-Level Approach", *Small Group Research*, 2004, Vol. 35, pp. 515-539.

[5] B. Barry, G.L. Stewart, "Composition, process and performance in self-managed groups: The role of personality", *Journal of Applied Psychology*, 1997, Vol. 82, pp. 62-78.

[6] M.R. Barrick, M.K. Mount, "The Big Five personality dimensions and job performance: A meta-analisys", *Personnel Psychology*, 1991, Vol. 44, pp. 1-26.

[7] M.R. Barrick, G.L. Stewart, M.J. Neubert, M.K. Mount, "Relating Member Ability and Personality to Work Team Processes and Team Effectiveness", *Journal of Applied Psychology*, 1998, Vol. 83, pp. 377-391.

[8] H.-L.Yang, J.-H. Tang, "Team structure and team performance in IS development: A social network perspective", *Information & Management*, 2004, Vol. 41, pp. 335-349.

[9] S. Faraj, L. Sproull, "Coordinating expertise in software development teams", *Management Science*, 2002, Vol. 46(12), pp. 1554-1568.

[10] W.S. Humphrey, M.D. Konrad, *Motivation and Process Improvement*, In: Software Process Modeling, Springer, 2005.

[11] Jr.P.T. Costa, R.R. McCrae, *NEO Personality Inventory*, Psychological Assessment Resources, 1992.

[12] E. Molleman, "The modalities of self-management: the "must", "may", "can" and "will" of local decision making", *The International Journal of Operations and Production Management*, 2000, Vol. 20, pp. 889-910.

[13] T. DeMarco, T. Lister, *Peopleware: Productive Projects and Teams* (2nd ed), Dorset House, New York, 1999.

[14] B.W. Boehm, C. Abts, W.A. Brown, S. Chulani, B.K. Clark, E. Horowitz, R. Madachy, D.J. Reifer, B. Steece, *Software Cost Estimation with COCOMO II*, Upper Saddle River: Prentice Hall PTR, 2000.

[15] R.H. Rutherford, "Using personality inventories to help form teams for software engineering class projects", *SIGCSE-Bulletin*, 2001, Vol. 33(3), pp. 76–76.

[16] R.P. Bostrom, K.M. Kaiser, "Personality differences within systems project teams: Implications for designing solving centers", *Proceedings of the Eighteenth Annual ACM SIGCPR Conference,* 1981, pp. 248–285.

[17] L.T. Hardiman, "Personality types and software engineers", *IEEE Computer*, 1997, Vol. 301(10), pp 10–10.

[18] K. White, R. Leifer, "Information systems development success: Perspectives from project team participants", *MIS Quaterly*, 1986, Vol. 10(3), pp. 215–23.

[19] G. Burdett, R-Y. Li, "A quantitative approach to the formation of workgroups", *Proceedings of the ACM SIGCPR Conference*, 1995, pp. 202–212.

[20] A. Zakarian, A. Kusiak, "Forming teams: An analytical approach", *IIE Transactions*, 1999, Vol. 31, pp. 85–97.

[21] W. Zuser, T. Grechening, "Reflecting skills and personality internally as means for team performance improvement", *Proceedings of the 16th Conference on Software Engineering Education and Training, IEEE Computer Society*, 2003.

[22] B. Tuckman, "Developmental sequence in small groups", *Psychological Bulletin*, 1965, Vol. 63, pp. 384–399.

[23] T.D. Cook, D.T. Campbell, *Quasi-Experimentation Design and Analysis Issues for the Field Settings*, Houghton Mifflin, Boston, MA, 1979.

[24] K. Beck, *Extreme Programming Explained: Embrace Change*, Addison-Wesley, Reading, 1999.

[25] D.L. Gladstein, "Groups in Context: A Model of Task Group Effectiveness", *Administrative Science Quarterly*, 1984, Vol. 29, pp. 499.

[26] J.P. Stokes, "Toward an understanding of cohesion in personal change groups", *International Journal of Group Psychotherapy*, 1983, Vol. 33, pp. 449-467.

[27] G. Van der Vegt, B. Emans, E. Van de Vliert, "Affective reactions to individual task interdependence in outcome interdependence groups", *Personnel Psychology*, 2001, Vol. 54, pp. 51-69.

Estimación basada en Escenarios Principales

José Cao, María Ochoa, Enrique Fernández, Hernán Merlino, Alejandro Hossian, Enrique Sierra, Eduardo Diez, Paola Britos, Ramón García-Martínez
Centro de Ingeniería del Software e Ingeniería del Conocimiento. Escuela de Postgrado. ITBA.
Laboratorio de Sistemas Inteligentes. Facultad de Ingeniería. Universidad de Buenos Aires.
Departamento Electrotecnia. Facultad de Ingeniería. Universidad Nacional del Comahue
hossi@ciudad.com.ar, esierra@uncoma.edu.a, rgm@itba.edu.ar

Resumen

La correcta estimación temprana de un proyecto de software es una tarea difícil o casi imposible. Sin embargo la mayor parte de los sistemas que son desarrollados por terceros requieren fijar un precio antes de contratarse el desarrollo. La estimación es necesaria para la definición de ese precio. El presente trabajo propone la estimación de casos de uso por comparación de sus escenarios con escenarios normalizados, explicando, además, la desventaja de otros métodos utilizados.

1. El problema de la estimación temprana

En el marco de la gestión de proyectos puede definirse la estimación como "el proceso que proporciona un valor a un conjunto de variables. Para la realización de un trabajo, dentro de un rango aceptable de tolerancia." [1]. Los problemas de la estimación son varios. Sabemos que estimar significa, de alguna forma, predecir el futuro, actividad con una incertidumbre que se refleja en la frase "rango aceptable de tolerancia" de la definición dada. Una de las formas de predecir el futuro es tomar en cuenta lo que sucedió en el pasado. En ese sentido, las técnicas de estimación se basan de una forma u otra en datos históricos y experiencias previas sobre elementos o subelementos similares al que se debe estimar. Surge de esto un primer problema, la identificación de elementos o subelementos similares o comparables a lo que debemos estimar. Ahora bien, la actividad de estimación no se hace una sola vez en el proyecto. A medida que se cuenta con más datos se hacen estimaciones más precisas que nos permiten una mejor planificación de lo que resta del proyecto. De todos los puntos en los cuales puede realizarse la estimación, cuando menos datos tenemos para hacerla es en el momento inicial, cuando todavía se está evaluando la factibilidad económica del proyecto. Desde el punto de vista del desarrollo y venta de software específico para terceros, esa estimación, que llamaremos temprana, es una de las más importantes porque fija el precio de contratación. Si nuestro objetivo es medir el software debemos establecer algún elemento o parámetro básico de estimación a usar en una métrica. Es decir, un elemento básico para representar el software, o parte de él, el cual deberá ser susceptible de asignársele algún valor para llegar a un número final que expresará en forma cuantitativa la medida de ese software. El objetivo de este trabajo es demostrar la conveniencia del uso de escenarios normalizados como elemento básico de comparación con escenarios reales, tanto para la estimación temprana como en cualquier otro punto del proceso.Por otra parte, para aclarar algún punto o enfatizar su factibilidad, nombraremos la experiencia de la empresa argentina Idea Factory Software (empresa evaluada CMM nivel 3), quién desarrolló un método de estimación basado en estos principios y que utiliza con éxito desde hace un año y medio.

2. Inconvenientes de la técnica de puntos de función

Una medida del tamaño del un sistema, de uso generalizado son los puntos de función. El concepto de punto de función fue presentado en 1979 (primera publicación) por Allan Albrecht refinándolo en 1984. El método intenta medir el tamaño de un sistema software. El presente trabajo no tendría sentido si pensáramos que los puntos de función solucionan el problema planteado.A los efectos de una estimación temprana los puntos de función tienen el inconveniente de asumir que se conocen los grupos de datos que utilizará el sistema y además asume que ese conocimiento es lo suficientemente amplio como para calificar esos grupos en complejidades baja, media y alta. En general en el período de preventa no se dispone de esos datos, especialmente de los datos internos al sistema. Ese es el primer problema, casi insuperable. Por otra parte, si se dispusiera de esa

información es complicado en tiempo y costo hacer el análisis necesario para calcular los puntos de función. Podría quizás utilizarse un método simplificado evaluando sólo las entradas, salidas y consultas. En ese caso utilizaríamos la información funcional disponible, aunque el modelo quedaría demasiado simple para representar la realidad. Por otra parte hay una consideración adicional. En la actualidad el análisis, diseño y desarrollo de sistemas se hace principalmente con metodologías orientadas a objeto. La técnica de los Casos de Uso, si bien no es privativa de la orientación a objetos, es claramente preponderante. El desarrollo del sistema es conducido por casos de uso [2] y los casos de uso se utilizan como unidades tanto de análisis, diseño, pruebas como de desarrollo. Si utilizáramos puntos de función deberíamos hacer un análisis (entradas, salidas, consultas, datos) que sólo se usa para esta técnica. Sería más eficiente hacer un análisis que además de servir para la estimación coincida con la metodología utilizada. Esto es importante si se tiene en cuenta que deberá reestimarse varias veces en el transcurso del proyecto para obtener las diferentes métricas de gestión, con lo cual no es eficiente hacer cada vez una conversión de una unidad de trabajo en otra (puntos de función a casos de uso). En pocas palabras, si nuestra unidad de trabajo es el caso de uso, es más natural contabilizar casos de uso en lugar de entradas, salidas, consultas y datos.

3. Los casos de uso como elemento básico inadecuado

3.1. Casos de uso como elemento básico – Método Use Case Points

Como ya dijimos en la sección anterior, dado que los casos de uso son la unidad rectora en la construcción de sistemas orientados a objetos, es natural considerar el caso de uso como un elemento o parámetro de básico apto para el proceso de estimación. Un ejemplo de este criterio es el difundido método Use Case Points desarrollado sobre el trabajo de Gustav Karner en 1993 [3] y que recrea de alguna forma el trabajo de Allen Albrecht sobre puntos de función. El método Use Case Points clasifica los casos de uso en simple, promedio y complejo con factores de peso 5, 10 y 15 respectivamente. La clasificación se hace en base al número de transacciones que contiene el caso de uso, 1 a 3 para simple, 4 a 7 para promedio y 8 ó más para complejos. Por otra parte se define una transacción como un evento que ocurre entre un actor y el sistema a ser modelado. A mi juicio, esta clasificación es totalmente inadecuada a los efectos de realizar una estimación.

El principal motivo de esto es que un caso de uso no tiene un tamaño determinado. En un ensayo para Rational Software, John Smith [4] considera como normal un promedio de 30 escenarios por caso de uso, tratando de ser conservador en el número. Un caso de uso de este tipo podría llegar a tener 30 transacciones. Si a cada escenario le correspondiera una transacción, este caso de uso tendría igual factor de peso que uno de 8 transacciones. Es decir el rango superior, 8 a infinito, da igual valor para 8 transacciones que para infinitas, lo cual es un inconveniente. Ahora bien, los inconvenientes persisten si se mejora la escala asignando puntos de caso de uso proporcionalmente al número de transacciones. En efecto la complejidad de un escenario puede ser tanto como la complejidad de otros 10, ó 20 ó el número que se quiera elegir y lo mismo vale para las transacciones. Por otra parte la agrupación de los requerimientos en casos de uso puede ser totalmente arbitraria haciendo que el método dé valores muy distintos para el mismo sistema. En ese sentido he visto un trabajo sobre el método de Use Case Points en donde para hacer un programa de altas, bajas, modificaciones y consultas resultó un esfuerzo de programación de 4 meses utilizando el lenguaje C++. Para un programa de altas, bajas, modificaciones y consultas, cuatro meses es a todas luces excesivo. En ese caso el método fue bien aplicado, sin embargo se consideró un caso de uso para el alta, otro para la baja, otro para la modificación y otro para la consulta. Hoy en día podemos decir, como explica Bittner [5] que semejante división es un error, que alta, baja, modificación y consulta no conforman cuatro casos de uso. Sin embargo en el libro Writing Effective Use Cases [6], quizás ya fuera de época, podemos leer que la opción de hacer un caso de uso para cada una de esas funciones, alta, baja, modificación y consulta, es una alternativa válida. Por otra parte hay quien, con criterio razonable, establece un caso de uso con el alta, y otro con la consulta, siendo baja y modificación extensiones de esta. Es decir, en algo tan simple y común como un programa de mantenimiento podemos considerar uno, dos o cuatro casos de uso, haciendo que nuestra estimación llegue casi hasta cuadruplicarse según la separación que hagamos. En conclusión, un método basado en casos de uso debería entonces tener en cuenta el número de transacciones o escenarios del mismo para determinar su peso. Sin embargo teniendo en cuenta esto no resolveríamos el problema de la complejidad que pueda tener el conjunto de transacciones o escenarios contenidos en cada caso de uso, es decir no es simplemente un problema de cantidad sino también de complejidad.

3.2 Las Clases del Sistema como alternativa a los Casos de Uso

En función de las consideraciones anteriores nos vemos tentados a abandonar el camino marcado por los casos de uso para buscar otras alternativas.

En ese sentido una unidad compatible con la orientación a objetos podría ser las clases involucradas en el sistema. La metodología española Métrica 3 [7] en su sección "Técnicas y prácticas" sabiamente elude el caso de uso como parámetro de estimación recomendado y plantea la posibilidad de utilizar el método Staffing Size. Define este método como "un conjunto de métricas para estimar el número de personas necesarias en un desarrollo Orientación a Objetos, y para determinar el tiempo de su participación en el mismo". Las unidades que utiliza son el número de clases clave y clases secundarias existentes en el modelo y toma en cuenta el lenguaje de programación utilizado. En este caso, al igual que los casos de uso, las clases clave pueden llevar asociadas funciones más o menos complejas y con más o menos operaciones a programar. En conclusión algunos de los inconvenientes indicados para los casos de uso subsisten si se utilizan las clases como parámetro de comparación.

3.3. Los Casos de Uso - Unidad de Agrupación

Como dijimos, los casos de uso son la unidad de trabajo en los métodos orientados a objeto que los utilizan. En este sentido es importante que la estimación también pueda expresarse en casos de uso. Si bien afirmamos en esta sección que los casos de uso no deben ser el parámetro básico para la estimación, eso no quita, que no siendo el parámetro básico, sí puedan ser una unidad de agrupación.

4. La falacia del tamaño independiente de la tecnología

4.1 El Tamaño y el Esfuerzo

Hasta este momento hemos hablado de estimación, pero, en lo posible, evitamos cuidadosamente aclarar si nos referimos a estimación de tamaño o de esfuerzo. No es posible seguir adelante sin hacer algunos comentarios al respecto, principalmente porque al hablar de estimación cualquier interlocutor educado en la teoría, pregunta si es estimación de tamaño o esfuerzo, y no acepta seguir hablando si no se hace esa aclaración.

Esta actitud cuasi dogmática, que se observa a menudo en los mejores profesionales, justamente por ser los mejores, se debe a que la posibilidad de cuantificar puramente el tamaño de un sistema, es un dogma que, como tal, sólo depende de la fe del que lo esgrime.

4.2 Un Sistema no es una mesa

En efecto, si hablamos de construir una mesa tenemos claro qué es el tamaño. Puede ser una mesa de 1m de altura, con una tabla de 1m por 2m. Podemos dar el volumen en m3 de la madera a utilizar o dar su peso esperado como medida del tamaño. Por otra parte el esfuerzo en construirla se medirá en horas. Evidentemente el esfuerzo será dependiente de la tecnología a utilizar. No es lo mismo hacer una mesa con sierra manual que con sierra eléctrica. Por otra parte la habilidad del carpintero y sus condiciones de trabajo también influirán en la cantidad de horas que insumirá la construcción. Entonces el tamaño es un atributo de la mesa y el esfuerzo depende del tamaño pero también de otros factores, tecnológicos, de habilidad etc. Este esquema simple se intenta utilizar en el desarrollo de sistemas. Se mide el tamaño, el cual se considera independiente de la tecnología como primer atributo del mismo, y en función de ese tamaño se calcula el esfuerzo para una determinada tecnología y condiciones de trabajo. El inconveniente comienza cuando intentamos medir el sistema para determinar su tamaño. Evidentemente ni los centímetros ni los kilogramos utilizados para la mesa son útiles para medir el tamaño de un sistema, especialmente si aún no fue construido. Es necesario entonces identificar una unidad de medida del tamaño del sistema y aquí se pierde la independencia de la tecnología y se desdibuja el puro concepto de tamaño. Las unidades que inicialmente se nos puedan ocurrir terminan siendo dependientes de la tecnología utilizada, líneas de código, tamaño en bytes etc. Podemos decir que el tamaño de un sistema está relacionado con la funcionalidad del mismo. En ese sentido nada parece más indicado que utilizar algo así como "puntos de función" para medir la funcionalidad. El más popular exponente de este sistema es el método de puntos de función de Albretch. De este método se afirma que mide tamaño independiente de la tecnología. El método dice, por ejemplo, que una entrada de complejidad media tiene 4 puntos de función y que a una consulta media también le corresponden 4 puntos de función. Ante esta afirmación surge una pregunta, ¿por qué la función de entrada de datos es equivalente funcionalmente a la de consulta? ¿Qué criterio se utilizó para establecer esa equivalencia entre estas dos

funciones y todas las demás que conforman el método?. Albretch estudió 24 proyectos de aplicaciones de negocios con un rango de tamaño desde 3000 a 318.000 líneas de código desarrolladas en DMS, PL/1 y COBOL. Podemos asumir, dado que no encontré una explicación, que en esos proyectos Albretch halló una relación entre entrada y consulta que le permite decir que la cantidad de líneas de código utilizadas para una entrada de complejidad media es igual a la cantidad de líneas de código de una consulta media. Sin embargo tenemos que reconocer que esa igualdad de líneas de código se da para los lenguajes analizados y no necesariamente para cualquiera. En efecto, si un lenguaje tiene facilidades para programar una consulta entonces esa relación ya no existe. Vemos entonces que el tamaño, "independiente de la tecnología" del método de puntos de función ya no es tan independiente de la tecnología, y por lo tanto no sería tamaño según la definición clásica, sino esfuerzo normalizado para una o más tecnologías.

4.3. Tamaño no fácilmente cuantificable

Todo esto se puede generalizar a cualquier tipo de métrica que pretenda medir tamaño funcional reduciendo a una unidad común los diferentes tipos de funciones. La única forma de definir puramente y estrictamente tamaño, independientemente de cualquier tecnología, es enumerar la cantidad de funciones de un mismo tipo que tiene un sistema, por ejemplo decir que el tamaño es 20 altas simples, 30 modificaciones complicadas, 12 listados simples, etc., pero al tratar de establecer una equivalencia entre un tipo función y otra, necesariamente se acude a nociones relacionadas con el esfuerzo de construcción de las mismas (codificación, diseño,análisis, etc.), ya que las únicas unidades de medida que tiene en común un alta con una modificación o un listado son las horas que se tarda en construirlas o también el tamaño físico (sean bytes o líneas de código o páginas de documentación) que tienen en una determinada tecnología o método. En conclusión, en el método que proponemos, cuando decidimos cuantificar escenarios estaremos hablando de "tamaño, en cierta medida, dependiente de la tecnología", que también podemos llamar "esfuerzo normalizado", es decir definido y medido definiendo como fijos y standard un conjunto determinado de factores tecnológicos. Como ejemplo de lo anterior en la implementación práctica de este método realizada por Idea Factory Software fue necesario realizar una distinta escala de valores para alguna tecnología en donde las relaciones de esfuerzo entre las funciones variaba notablemente con respecto a las otras.

5. Escenarios principales como elemento básico adecuado

5.1. Tres Problemas: Normalización, Cuantificación y Extrapolación

En la sección 3 de este trabajo hemos postulado que los casos de uso no son un elemento básico adecuado para la estimación. Por otro lado son la unidad de trabajo natural en un desarrollo de software orientado a objetos. Debemos encontrar un elemento básico de estimación que salve los inconvenientes descriptos anteriormente, que sea agrupable por caso de uso, para poder controlar la evolución de cada caso de uso, y a la vez que implique un esfuerzo de análisis que luego pueda ser aprovechado. El tamaño de un sistema depende de su funcionalidad. Funcionalidad es un término que de alguna manera significa lo que el sistema puede hacer, en suma, el conjunto de sus funciones. Como expusimos en el capítulo anterior, cuantificar el tamaño implica algunos inconvenientes y asunciones que deben hacerse. En ese esquema hay una solución para llegar a la estimación si se resuelven tres problemas principales:

a) ¿Cómo clasificar en forma normalizada las funciones ("funcionalidad") de un sistema? **(normalización)**

b) ¿Qué valores numéricos comparativos se deben asignar a cada función normalizada? **(cuantificación)**

c) ¿Qué relación hay entre una unidad de medida de cada función (item b) y la unidad de esfuerzo de la actividad que se quiere estimar. **(extrapolación)**

5.2 Los Escenarios como elemento básico de estimación

Ahora bien, si queremos tener un esquema de estimación compatible con nuestra metodología orientada a objetos, deberíamos buscar en esa metodología qué elemento será la unidad básica de estimación en nuestro sistema.

En orientación a objetos las funciones se traducen en operaciones contenidas por las clases. Pero no se llegan a identificar las operaciones hasta bien avanzado el proyecto, por lo cual no podrían usarse como elemento de estimación en una etapa temprana. Por otro lado su número y granularidad es tal que se

harían casi inmanejables. El nivel superior a la operación, en cuanto a elementos funcionales de la orientación a objetos, es el escenario, y el nivel superior a ese es el caso de uso. Hemos hecho consideraciones por las cuales no consideramos apropiado la utilización de casos de uso, por lo cual deberemos analizar el uso de escenarios. Formalmente un escenario representa una forma de uso del sistema, no es exactamente una función, pero sí representa la funcionalidad del sistema. Por otra parte, si sabemos lo que el sistema debe hacer, entonces podemos saber cuáles serían sus escenarios principales. Esta última afirmación puede ponerse en duda a priori, sin embargo la experiencia práctica realizada por Idea Factory Software tuvo repetidamente éxito en demostrar este punto. Ahora bien, es evidente que la cantidad de escenarios principales podría servirnos para el cálculo del tamaño/esfuerzo relacionado con un desarrollo, pero debería considerarse la complejidad del escenario para que esa medida fuera proporcional a la funcionalidad. Es necesario aquí detenerse para aclarar algo sobre los escenarios principales, dado que la teoría y la práctica son un tanto ambiguas al respecto. Por un lado se puede considerar que un caso de uso tiene uno y sólo un escenario principal y luego escenarios alternativos. Es algo que inicialmente tiene sentido y parte de la teoría coincide con esta idea. Por otro lado, si como dijimos en la sección 3, se tiende a que las funciones de mantenimiento de una clase, alta – baja – modificación y consulta, deban agruparse en un sólo caso de uso ¿podemos decir que hay un escenario principal , alta por ejemplo, y que consulta es una alternativa?. Esto sería raro dado que el valor agregado obtenido por el usuario difiere en cada escenario y ambos escenarios no tienen casi ningún paso en común. Es por eso que consideramos la postura que contempla que un caso de uso puede tener más de un escenario principal, cada uno con sus eventuales escenarios alternativos. Esta postura está sustentada en la teoría más pura si consideramos que en el libro básico de UML [8] dice que “Para cada caso de uso, usted encontrará escenarios primarios (los cuales definen secuencias esenciales) y escenarios secundarios (los cuales definen secuencias alternativas)”, declaración esta que contempla la existencia de más de un escenario principal, o primario, por caso de uso. En función de todo lo expuesto vemos que los escenarios principales comienzan a cumplir las características necesarias para ser elementos básicos para la estimación de un sistema. En efecto, no dependen de cómo se agrupan los escenarios en los casos de uso, pero permiten expresar la estimación en casos de uso. Representan la funcionalidad del sistema, con información disponible en etapas tempranas del desarrollo y además la clasificación del sistema en escenarios no es trabajo adicional ya que de todos modos debe hacerse para un análisis orientado a objeto. Por otra parte son un elemento fácilmente identificable, apto para ser valorado numéricamente y apto para ser refinado a medida que se obtiene mayor información.

5.3. Normalización

Corresponde ahora resolver el tema de la clasificación normalizada de esos escenarios. Para esto será necesario poder clasificar los escenarios principales en una forma standard y repetible. Una de las soluciones es entonces encontrar una lista de escenarios típicos, conocidos por todos los estimadores, y que de alguna forma puedan asimilarse a (o más bien compararse con) cualquier escenario que encontremos en la realidad. Además, para una mejor clasificación relacionada con la complejidad, cada escenario de esa lista podrá tener subtipos definidos en función de distintos atributos de los mismos. Por ejemplo puede existir un escenario de tipo alta con subtipo simple (en función del número aproximado de campos involucrados, por ejemplo) , subtipo normal y subtipo cabecera - detalle. Para cada tipo - subtipo deberá especificarse el flujo normal y cuáles alternativos y excepciones se consideran standard, a fin de poder compararlo con el escenario que corresponda de la vida real. Ahora bien, ¿es posible hacer una lista de escenarios normalizados tal que cualquier escenario de un sistema real pueda asimilarse a un escenario de la lista?. Basados en el método utilizado por Idea Factory Software podemos decir que la respuesta es sí. En la implementación realizada por esta empresa se utiliza una lista de 11 escenarios normalizados que cubren todos los escenarios posibles, al menos en sistemas de gestión administrativa, contable y financiera en donde el método fue aplicado.

5.4. Desnormalizando la normalización - Complejidades y Facilidades adicionales

Por otra parte, es evidente que en el mundo real los escenarios difieren en mayor o menor medida de los escenarios normalizados. Si no se quiere tener una lista extensa de escenarios normalizados podemos buscar un elemento de ajuste para esos escenarios. Por ejemplo, nosotros podemos decir que un determinado escenario se parece bastante a la definición de un escenario que llamamos “Alta” con subtipo “simple”, pero sin embargo, desde el principio identificamos que

ese escenario principal tiene una gran cantidad de escenarios alternativos. Entonces se toma en cuenta una "complejidad" que represente esa característica y que sume puntos de escenario, quizás en función de la cantidad aproximada de escenarios alternativos que se estiman para el escenario real. Por otra parte puede existir una condición que haga que un escenario de la vida real sea más fácil que uno normalizado (por reuso de componentes, por ejemplo). En ese caso se involucraría una "facilidad" que reste puntos de escenario. En pocas palabras las complejidades/facilidades cubren la diferencia que pueda existir entre el escenario de la vida real y el, o los, escenarios normalizados utilizados para representarlo. Sabemos que la estimación es necesariamente un proceso que puede tener grandes variaciones y entonces es posible que los ajustes a los escenarios normalizados por complejidades o facilidades se consideren irrelevantes. En todo caso sólo deben usarse para cubrir diferencias significativas entre el escenario real y el normalizado. Si se quieren usar complejidades y facilidades, es evidente que, además de la lista de escenarios normalizados, deberá identificarse una lista de complejidades o facilidades más o menos comunes y cuantificarlas. Si bien es fácil hacer una lista corta de escenarios base , no es tan fácil cubrir todo el espectro de las complejidades y facilidades, por lo cual debería dejarse a criterio del estimador crear y cuantificar alguna complejidad/facilidad que pueda agregarse en el momento de la estimación. El método a desarrollar tendría entonces una fase más o menos automática y necesitará de algún esfuerzo específico para estimar alguna complejidad/facilidad. Afortunadamente los sistemas de gestión no nos dan excesivas sorpresas en cuanto a sus requisitos por lo cual, si la lista de escenarios normalizados está bien armada, no resulta necesaria la inclusión de muchas complejidades/facilidades. En la experiencia en Idea Factory Software como promedio por cada 40 escenarios tipo sólo se necesitó 1 facilidad/complejidad. Por otra parte este trabajo propone, a fin de cuentas, estimar por comparación; en nuestra lista de escenarios normalizados tratamos de tener escenarios suficientemente representativos de la realidad, pero nada quita que un escenario de la vida real, absolutamente atípico, sea comparable a uno normalizado, no por su funcionalidad, sino por su complejidad similar. Finalmente es necesario hacer una observación importante. Sabemos que los escenarios de modificación y baja pueden ser expresados como escenarios principales del caso de uso o como alternativos del escenario de consulta. Si no se toman en cuenta complejidades y facilidades, y esos escenarios son diseñados como alternativos, entonces no estaríamos estimando las bajas y las modificaciones, dado que el método sólo contempla escenarios principales. Esto es a todas luces un despropósito. En efecto, altas, bajas, modificaciones y consultas cubren la mayor parte cualquier sistema de gestión y deben ser incluidos sea como escenario normalizado o como complejidad standard, la exclusión de los escenarios alternativos en el método se debe principalmente a que habitualmente no se conoce ese nivel de detalle en la estimación temprana.

5.5. Cuantificación

Una vez resuelto el problema de la normalización, la cuantificación implica asignarle valores, cuya unidad podemos llamar puntos de escenario, a cada escenario normalizado tipo-subtipo. Los valores deberán asignarse en función de mediciones previas realizadas, o si estas faltan, en función de entrevistas con distintos profesionales por especialidad, teniéndose la dispersión de resultados como una medida de la confiabilidad de estos datos. En el caso de utilizarse un sistema de complejidades/facilidades, si cada escenario normalizado tiene un peso en "puntos de escenario" deberán identificarse además las distintas complejidades y facilidades adicionales que se pueden dar en cada caso y que, debidamente valorizadas, incidirán en la cantidad de "puntos de escenario" a asignar al escenario real. La naturaleza del valor que asignemos a cada escenario dependerá de lo que queremos medir. Si queremos medir esfuerzo asignaremos valores relacionados con el tiempo de desarrollo que llevaría cada escenario base. Si queremos medir tiempo de análisis asignaremos valores relacionados con el tiempo de análisis que llevaría cada escenario base. Y así podríamos definir para un escenario tipo puntos de escenario de desarrollo, puntos de escenario de análisis, puntos de escenario de testing etc. Sin embargo este procedimiento es trabajoso, y dada la poca exactitud de la actividad de estimación en general, se justifica utilizar una actividad única y extrapolarla al resto de las actividades. Ese problema es el que llamamos "extrapolación".

5.6. Extrapolación

Como dijimos, a los efectos de simplificación es conveniente entonces adoptar una actividad única para la cuantificación de los escenarios tipo y a partir de allí utilizar esa escala para las otras actividades. Tendremos así:

- Horas de codificación = K x suma de puntos de escenarios tipo
- Horas de análisis = L x suma de puntos de escenarios tipo
- Horas de diseño = M x suma de puntos de escenarios tipo
- Horas de testing = N x suma de puntos de escenarios tipo

Donde los "puntos de escenario" fueron asignados a cada escenario teniendo en cuenta su peso relativo en esfuerzo para una determinada actividad y K, L, M y N los factores de conversión en horas. La práctica realizada por Idea Factory Software demostró que, dado el carácter aproximado de la estimación, esta simplificación representa suficientemente la realidad, aunque implique eventualmente tener un escala de valores distinta para alguna tecnología que se desvíe de la relación más usual entre los distintos tipos-subtipos. La actividad utilizada, y que recomendamos, fue el tiempo de codificación de cada escenario. Es una tentación elegir el tiempo de análisis como actividad base ya que depende sólo de la metodología de análisis y no de la tecnología. En ese sentido estaríamos cerca nuevamente del concepto de tamaño. Sin embargo los valores de análisis resultan más dispersos que los de codificación quizás porque el tiempo de análisis depende también de la eficiencia del usuario final para expresar y validar los requerimientos y de la complejidad del problema. Si tenemos en cuenta además que la codificación es la actividad que lleva mayor tiempo (relación de 3 a 1 en J2EE, por ejemplo) la elección de la codificación como actividad para cuantificar la escala resulta la más ajustada a la realidad.

5.7. Comparación de un método basado en escenarios principales vs. uno basado en casos de uso

En esta sección desarrollaremos un ejemplo que permite ver la sensibilidad de un método basado en escenarios principales con respecto al Use Case Points [3]. Tomemos para el ejemplo un caso de uso con los escenarios principales de alta, consulta y modificación (en adelante ACM) sobre una entidad simple. Consideramos que no se contempla la baja física porque la misma es lógica y es resultado de una modificación. Con el método Use Case Points tendríamos:

3 transacciones → caso de uso simple → Factor de peso en Use Case Points = 5.

Consideremos ahora la necesidad de hacer una baja física y entonces se agrega ese escenario al caso de uso (en adelante ACMB para el caso de uso con escenario de baja) tenemos entonces:

4 transacciones →caso de uso medio → Factor de peso en Use Case Points = 10.

El aumento de peso por agregar el escenario de baja es de 100%. " Baja" es el escenario que menos esfuerzo lleva de los cuatro, pero su inclusión duplicó el peso. Esto demuestra que al método UCP le falta, al menos, suficiente granularidad para medir correctamente este tipo de diferencias. Probemos ahora con un método basado en escenarios principales. La tabla parcial para los escenarios elegidos, para tecnologías Java y .Net (tienen iguales relaciones relativas entre tipos de escenario) es:

Tabla (parcial) de escenarios con valoración de PE para tecnologías J2EE y .NET	
Escenario - Complejidad	**Puntos de Escenario (PE)**
Escenario "Alta" – "Simple"	+10
Escenario "Consulta" – "Simple"	+22
Escenario "Modificación" – "Simple"	+6
Escenario "Baja" – "Simple"	+4

Para el caso de uso ACM el método basado en escenarios principales da :

Alta Simple + Consulta Simple"+ Modificación Simple = 10PE + 22PE + 6PE = 38 PE

Si agregamos la baja para formar el caso de uso ACMB tenemos:

Alta Simple+Consulta Simple + Modificación Simple + Baja Simple =1

= 0PE+22PE + 6PE + 4 PE = 42PE

El aumento por agregar el escenario de baja es de 10,5%, cifra que representa el aumento de esfuerzo real medido para escenarios de baja simple. Demostrando así que el uso de escenarios principales es sensible y proporcional a los pequeños cambios. Veamos qué pasa con la complejidad. Consideremos ahora que la entidad sobre la que se ejecutan el alta, baja, modificación y consulta es una entidad compuesta que representa una relación de agregación (tipo cabecera-detalle). Para el caso de uso ACMB en el método Use Case Points no hay cambio evidente dado que a pesar de la mayor complejidad del caso de uso sigue siendo complejidad media porque el actor sigue teniendo 4 transacciones, es decir este tipo de cambio de complejidad no es detectado Para el método basado en escenarios principales la tabla de los escenarios involucrados es:

Tabla (parcial) de escenarios con valoración de PE para tecnologías J2EE y .NET	
Escenario - Complejidad	**Puntos de Escenario (PE)**
Escenario "Alta" – "Cabecera – detalle"	+28
Escenario "Consulta" – "Cabecera – detalle"	+25
Escenario "Modificación" – "Cabecera – detalle"	+10
Escenario "Baja" – "Cascada"	+5

El mayor peso del escenario de "alta" con respecto al de "consulta" se debe a que el escenario tipo "alta"

incluye las pantallas y navegación que luego se reutilizará en los otros escenarios. En este caso tendremos :

Escenarios Cabecera-detalle de Alta"+ Consulta + Modificación + Baja =

= 28 + 25 +10 + 5 = 68 PE

Vemos que el aumento en complejidad fue tenido en cuenta por el método e incrementó los puntos de escenario en un 61,9%, cifra que concuerda con las mediciones realizadas. Podemos concluir entonces que el método Use Case Points tiene algunos problemas de sensibilidad cuantitativos (cantidad de escenarios) y cualitativos (complejidad) y que un método basado en escenarios principales resuelve esos problemas. Por otra parte sus resultados no dependen de la forma en que los escenarios son agrupados en los casos de uso.

Finalmente, y asumiendo para simplificar que no hay modificaciones de ambiente ni de complejidad técnica general, podemos obtener esfuerzo para varias actividades extrapolando:

Codificación/prueba unitaria =	0,7 h/PE x Σ de PE =	0,7 h/PE x 68 PE = 47hs
Análisis =	0,24 h/PE x Σ de PE =	0,24 h/PE x 68 PE = 16hs
Diseño =	0,12 h/PE x Σ de PE=	0,12 h/PE x 68 PE = 8hs
Testing (planif.y ejecución) =	0,12 h/PE x Σ de PE =	0,12 h/PE x 68 PE= 8hs
Corrección de defectos =	0,12 h/PE x Σ de PE =	0,12 h/PE x 68 PE= 8hs

Donde los coeficientes usados para codificación y prueba unitaria, corrección de defectos y diseño se corresponden con un determinado lenguaje (Java) y arquitectura específica, y los coeficientes de testing y análisis se corresponden con un método y documentación evaluados CMM nivel 3.

6. Conclusiones

Todo lo expuesto en este trabajo puede resumirse en las siguientes conclusiones:

[a] Los casos de uso no son un elemento básico apto para una estimación razonablemente fina de las actividades de desarrollo de sistemas.

[b] Los escenarios principales son un elemento básico apto para estimar las actividades de desarrollo de sistemas en general y orientados a objeto en particular.

[c] Un método para dimensionar un sistema puede basarse en una comparación entre los escenarios del sistema real a construir y una lista de escenarios normalizados. A los efectos de lograr una estimación más fina, cada tipo de escenario normalizado puede ser dividido en subtipos teniendo en cuenta algún atributo del escenario relacionado con la complejidad de su construcción o análisis. Si se necesitara mayor fineza en la estimación pueden agregarse complejidades o facilidades, normalizadas o ad hoc.

[d] El concepto de tamaño no es tan claro en la ingeniería de software como en otras disciplinas de la ingeniería.

[e] En función de la experiencia de la empresa Idea Factory Software utilizando estos principios puede afirmarse la factibilidad técnica y económica de su aplicación.

7. Referencias

[1] Ana Ma. Moreno Sánchez Capuchino Módulo I *Control y gestión de proyectos software -– Unidad 4: Estimación de Proyectos Software*. Carpetas de la Carrera de Postgrado en Ingeniería de Software. Imprenta del Instituto Tecnológico de Buenos Aires.

[2] Ivar Jacobson, Grady Booch, James Rumbaugh. 2000. *El Proceso Unificado de Desarrollo de Software* Editorial Pearson Educación. ISBN: 84-7829-036-2.

[3] Gustav Karner. 2001. *Use Case Points - Resource Estimation for Objectory Projects*. Objective Systems SF AB

[4] John Smith. 1999. *The Estimation of Effort Based on Use Cases*. Rational Software White Paper.

[5] Kurt Bittner. 2001. *Why Use Cases Are Not "Functions"*. Rational Software.

[6] Alistair Cockburn**.** 2000. *Writing Effective Use Cases*. Addison-Wesley.

[7] Ministerio de Adm. Públicas. 1994. Metodología Métrica Versión 3 – Técnicas y prácticas - España

[8] Ivar Jacobson, Grady Booch, James Rumbaugh. 2002. *The Unified Modeling Language User Guide* - Editorial Addison – Wesley. ISBN: 0201571684.

RevisionCASE, herramienta para Gestionar Revisiones a Proyectos de Software empleando Razonamiento Basado en Casos.

MSc. Martha Delgado Dapena, Dra. Sofía Álvarez Cárdenas, Ing. Josué Carralero Iznaga, Ing. Javier Travieso Arencibia, Ing. Iren Lorenzo Fonseca, Dr. Alejandro Rosete Suárez.
Centro de Estudios de Ingeniería de Sistemas (CEIS), Instituto Superior Politécnico "José Antonio Echevarría" (CUJAE), Habana, Cuba.
{marta, jcarralero, jtravieso, ilorenzo, rosete}@ceis.cujae.edu.cu, sofiaalvarez48@gmail.com

Resumen

El resultado final de un proyecto de software es un producto que toma forma a lo largo del desarrollo del proyecto. La calidad del producto final, está estrechamente ligada a la calidad del Proceso de Desarrollo de Software, entre otros aspectos. Establecer un adecuado Sistema de Aseguramiento de Calidad que contribuya a detectar los defectos en las etapas tempranas del proceso de Desarrollo de Software contribuye considerablemente a elevar la calidad de los productos resultantes.

Una parte importante de este Sistema lo constituyen las Revisiones, que se realizan en los diferentes momentos del ciclo de desarrollo del proyecto. Con ellas se detectan defectos que pueden ser corregidos oportunamente, de forma tal que el producto final tenga la calidad necesaria.

En este trabajo se presenta un paquete de herramientas automatizadas para la gestión de revisiones a los proyectos de software, que ayudan a los revisores y desarrolladores a gestionar las revisiones realizadas a los proyectos. Una de las herramientas expuestas implementa técnicas de Razonamiento Basado en Casos (RBC) para asistir a los especialistas en la detección de defectos, para lo cual acumula la experiencia de detección de defectos en una Base de Conocimientos y la emplea en la ejecución de nuevas revisiones.

1. Introducción

El tema de la calidad del software tiene gran actualidad en el mundo. Sin embargo, aunque las empresas dedican grandes recursos a la adopción o definición de estándares de calidad, los resultados alcanzados no cubren las expectativas, ya que la productividad es baja, la cantidad real de recursos a consumir es casi impredecible y el resultado casi nunca tiene la calidad y profesionalidad requerida.

Varios trabajos reflejan la importancia de establecer un Proceso de Revisión en las empresas de software [3, 4, 15, 16], sustentado en que las dos terceras partes de los defectos de los sistemas son el resultado de errores cometidos en etapas tempranas del desarrollo del proyecto, por lo que se hace necesario prevenir los defectos o detectarlos en esas etapas [2,14].

MGRSoft (Modelo de Gestión de revisiones Basado en Casos) es un modelo para la Gestión de Revisiones Basado en Casos que consta de un Sistema de Procesos que contempla procedimientos, roles y métricas y que se basa en la utilización de la experiencia acumulada por la empresa en materia de revisiones que es almacenada en una Base de Conocimientos que posee toda la información referente al proceso de revisiones. Además, se incluye un conjunto de Herramientas Automatizadas integradas en un paquete que permite gestionar dicho conocimiento en los dos procesos definidos dentro del modelo.

El Sistema de Procesos incluye dos procesos: el de nivel estratégico "Gestión de Revisiones" y el clave "Revisiones de Proyectos Específicos". Este modelo está sustentado en la concepción de los estándares internacionales [5, 6, 7, 8, 9, 10, 11, 16] que consideran que deben existir procesos estratégicos a nivel de la organización que es donde se dictan las políticas y estrategias generales y se proponen las mejoras a los procesos claves y en un segundo nivel deben existir procesos claves donde se ejecutan las actividades que garantizan el cumplimiento de los objetivos de calidad de los procesos estratégicos, desde el prisma de cada proceso clave particular.

La Base de Conocimientos contiene la información referente a las revisiones que permite contar con la experiencia acumulada en este proceso para que pueda ser utilizada en el desarrollo y revisión de nuevos proyectos.

Los procesos definidos están soportados por un paquete de herramientas automatizadas integradas (RevisionCASE), que gestionan la Base de Conocimientos.

2. Herramientas automatizadas para MGRSoft

RevisionCASE, que integra la gestión de las actividades descritas en el modelo con un asistente para ayudar en la detección de los defectos, Asistente para Revisiones a Proyectos aprovechando la experiencia Anterior (ARPA). Cada una de ellas soporta diferentes actividades dentro del modelo, pero intercambiando información contenida en los repositorios como parte de MGRSoft.

RevisionCASE, paquete de herramientas que soporta los subprocesos de MGRSoft, está compuesto por seis herramientas que cubren los subprocesos del modelo y que son descritas a continuación:

1. Administración de Calidad, se definen las políticas que seguirá la empresa para la ejecución de las revisiones, entre las que están: las metodologías a seguir según el tipo de proyecto, la definición de usuarios y roles dentro de la gestión de revisiones en la empresa, entre otros. Además se define el Plan de Aseguramiento de Calidad para cada proyecto en ejecución. En la figura 1 se presenta una pantalla donde se puede observar parte de la información que se maneja en la aplicación.
2. Planificación de Revisiones Específicas, se planifican las actividades para cada miembro del equipo de revisión, así como las listas de chequeo a emplear y los elementos de configuración correspondientes. En la figura 2 se presenta la información que se maneja en la aplicación durante la planificación de las actividades asignadas a un miembro del equipo de revisión.
3. Registro de Defectos, cada revisor registra los defectos encontrados en los elementos de configuración revisados y se le da seguimiento a todas las fases de las revisiones específicas del proyecto. En la figura 3 se muestra el registro de defectos que debe llenar cada miembro del equipo de revisión.
4. ARPA, Asistente para Revisiones a Proyectos aprovechando la experiencia Anterior. Es empleada como asistente en la herramienta de Registro de Defectos a una revisión particular. Utiliza el RBC para hacer una propuesta a los revisores de un conjunto de defectos que están presentes en el producto que se revisa, en la fase correspondiente. Además muestra un conjunto de defectos que han sido encontrados en otros proyectos revisados anteriormente lo que puede ser utilizado también por el equipo de desarrollo para prevenir la ocurrencia de dichos defectos.

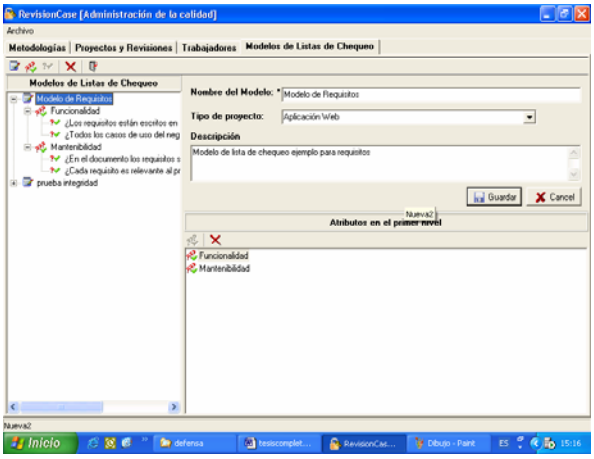


Figura 1. RevisionCASE: Administración de Calidad.

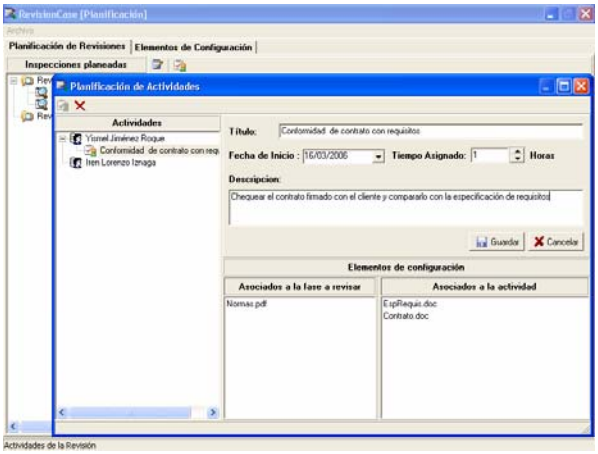


Figura 2. RevisionCASE: Planificación de Revisiones.

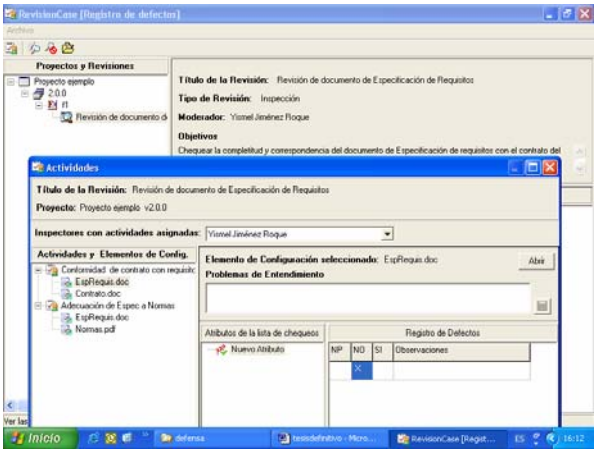


Figura 3. RevisionCASE: Registro de defectos.

5. Documentación y Métricas, se muestran los resultados de las métricas descritas en el capítulo 2 facilitándose el análisis por parte de los directivos de la empresa y de todos los involucrados en el proceso de revisión (Ver figura 4).
6. Seguimiento, se automatiza el subproceso de Seguimiento y Control de las Revisiones y el subproceso de Evaluación y Control del Proceso de Revisiones. Con ella es posible controlar la solución que se le da a cada defecto encontrado y la estrategia de solución seguida en cada caso, lo que permite que dicha estrategia pueda ser empleada en la solución de nuevos defectos que aparezcan en proyectos con características similares. Se identifican las causas de los defectos, lo que permitirá trabajar en la prevención de defectos en futuros desarrollos.

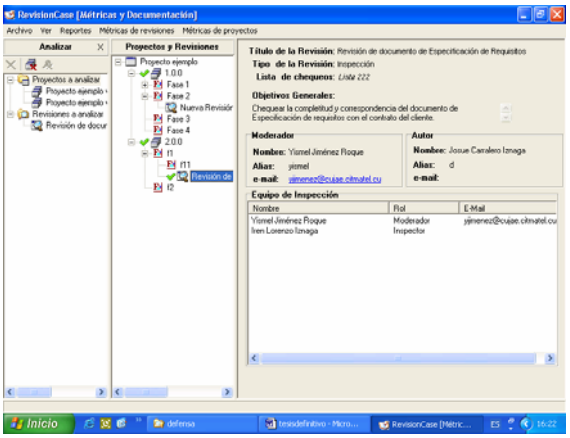


Figura 4. RevisionCASE: Documentación y Métricas.

El uso de las herramientas incluidas en RevisionCASE garantiza:

A los líderes de revisión:
- ✓ Prevenir el acceso no autorizado a los registros de defectos de los revisores hasta tanto no esté cerrada la revisión.
- ✓ Coordinar, dar seguimiento y dirigir las actividades asociadas a las revisiones específicas.

A los directores de calidad de los proyectos:
- ✓ Planificar y dar seguimiento a las revisiones del proyecto incluidas en el Plan de aseguramiento de Calidad del Proyecto y trazar Plan de Acciones correspondiente.

A los líderes de los proyectos:
- ✓ Analizar los resultados de las revisiones al proyecto y trazar el Plan de Acciones Correctivas correspondiente.

A los desarrolladores:
- ✓ Conocer qué ha pasado con cada elemento de configuración revisado.
- ✓ Conocer rápidamente las acciones correctivas asignadas para su ejecución.

A los directivos de la organización:
- ✓ Posibilidad de revisar y actualizar la planificación de revisiones rápidamente.
- ✓ Conocimiento de toda la información referente a la ejecución y resultados de las revisiones en cada uno de los proyectos; así como el desempeño de los miembros del equipo de revisión.

3. Asistente para Revisiones a Proyectos aprovechando la experiencia Anterior

ARPA parte de los requisitos candidatos o necesidades de los usuarios y de la solución que dieron los desarrolladores, es esta última la información que se somete a revisión por parte de los inspectores. El asistente desarrollado cuenta en la actualidad con algoritmos para asistir la detección de defectos en la etapa de Requisitos, pero puede ser extendido a otras etapas de desarrollo de proyectos incluyendo los algoritmos correspondientes en la fase de Adaptación del Sistema de Razonamiento Basado en Casos [1, 12, 13]. El sistema tiene tres tipos de usuarios:

1. Miembro del equipo de revisión o revisor: En este caso la información requerida por el sistema es la especificación completa que se quiere Revisar, que incluye el listado de Requisitos del Proyecto y su Modelo de Casos de Uso, representado en un Diagrama de Casos de Uso (Ver figura 5). La respuesta del sistema para este usuario está compuesta por un listado de defectos potenciales, encontrados en el proyecto que se está revisando, a partir de una comparación con una solución propuesta que considera la experiencia acumulada en la Base de Casos, y una propuesta de solución a estos defectos, materializada en un diagrama de casos de uso y una descripción de los requisitos (Ver figura 6).

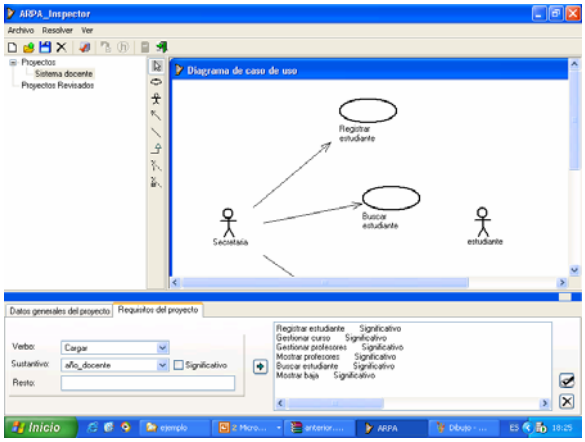


Figura 5. Información suministrada por el revisor en ARPA.

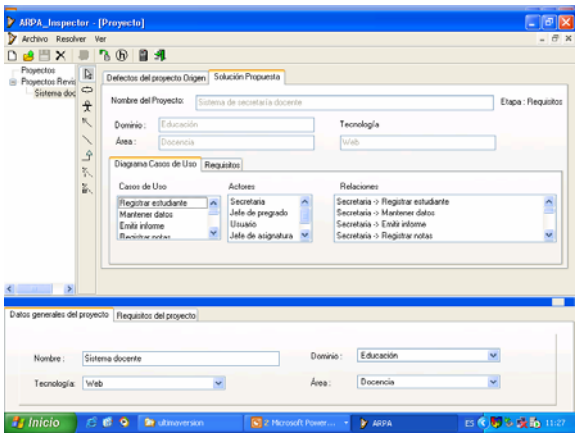


Figura 6. Propuesta de solución ofrecida por ARPA al revisor.

2. Miembro del equipo de desarrollo o desarrollador: La información a suministrar al sistema por el usuario es únicamente la lista de requisitos preliminares y el sistema de RBC devolverá al usuario un listado completo de Requisitos con su Modelo de Casos de Uso del sistema. Esta información se obtiene a partir de los casos más parecidos almacenados en la base de casos.
3. Administrador de la Base de Casos: Las actividades asociadas a este usuario son las de mantenimiento de la Base de Casos. La inserción de los casos a la Base sólo puede ser realizada desde el módulo de administración, puesto que el administrador es el único con permiso para hacer variaciones en los datos que maneja la herramienta, a través de los botones señalados en los óvalos de la figura 7. No obstante, el resto de los usuarios pueden hacer propuestas que serán revisadas por el administrador para su inclusión o no en la Base de Casos. Estas propuestas aparecen como "Proyectos pendientes a aprobar" y aparecen señalados con un óvalo en la figura 8.

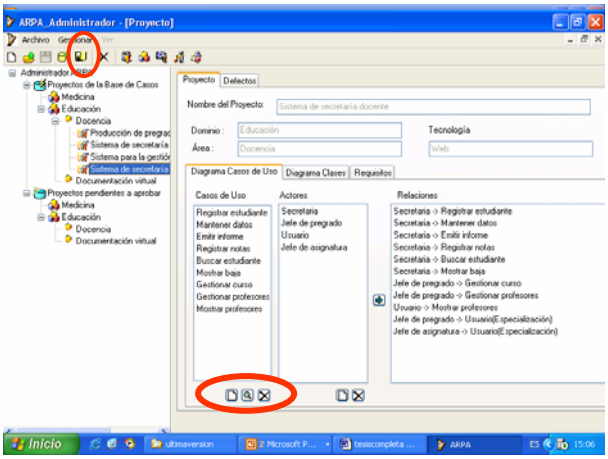


Figura 7. Posibilidad de modificación e inserción de los proyectos y los defectos detectados en la Base de Casos.

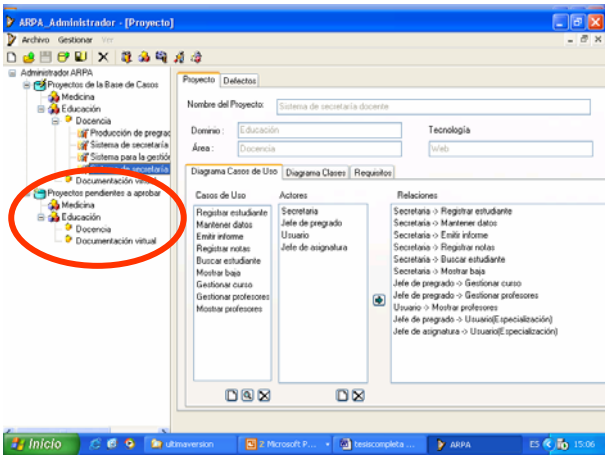


Figura 8. Análisis y aprobación de casos propuestos por revisores y desarrolladores.

A continuación se muestra un ejemplo de proyecto revisado con ARPA.

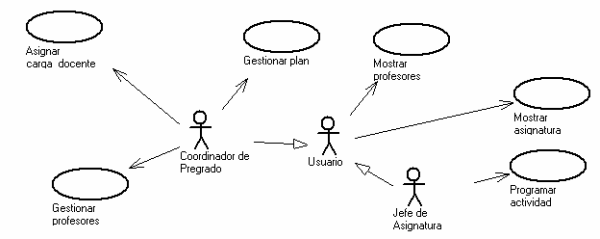
Ejemplo de información de la Base de Casos empleada por ARPA

Nombre del proyecto: Gestión del área de pregrado.
Dominio: Educación
Área: Pregrado
Requisitos preliminares:

1. Gestionar profesores.
2. Gestionar plan metodológico.
3. Asignar carga.

4. Programar actividad.
5. Mostrar asignaturas.
6. Mostrar profesores.

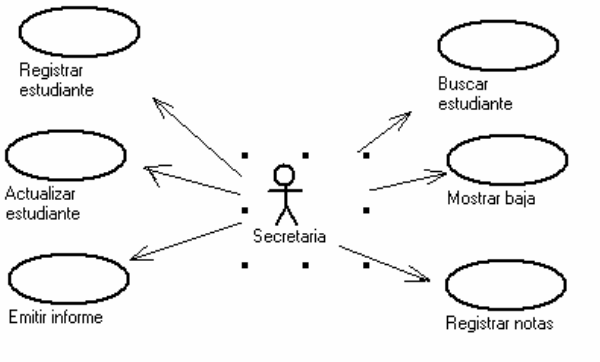
Diagrama de Casos de Uso del Sistema:



Nombre del proyecto: Secretaría Docente.
Dominio: Educación
Área: Pregrado
Requisitos preliminares:

1. Actualizar estudiantes.
2. Reportar promoción.
3. Registrar nota.
4. Registrar estudiante.
5. Buscar estudiante.
6. Mostrar Bajas.

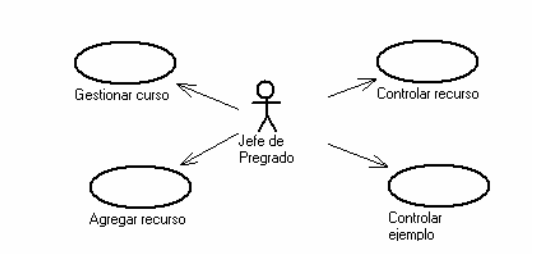
Diagrama de Casos de Uso del Sistema:



Nombre del proyecto: Producción de cursos a distancia.
Dominio: Educación
Área: Pregrado
Requisitos preliminares:

1. Gestionar curso.
2. Agregar recurso.
3. Controlar recurso.
4. Controlar fecha.

Diagrama de Casos de Uso del Sistema:

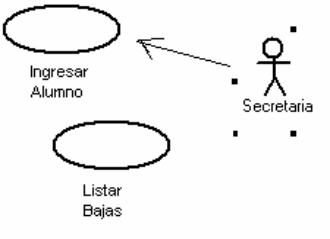


Ejemplo de proyecto a revisar

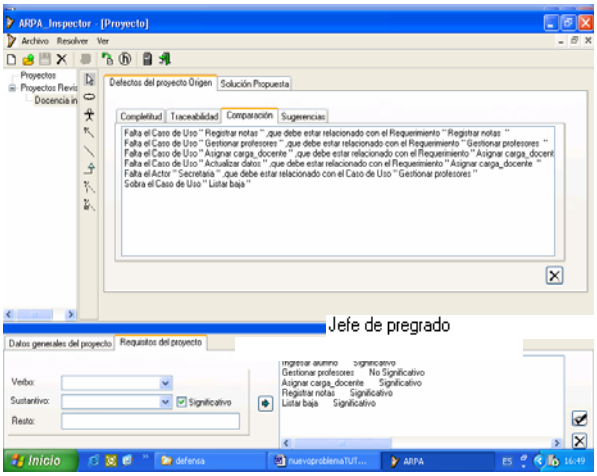
Nombre del proyecto: Docencia integral.
Dominio: Educación
Área: Pregrado
Requisitos preliminares:

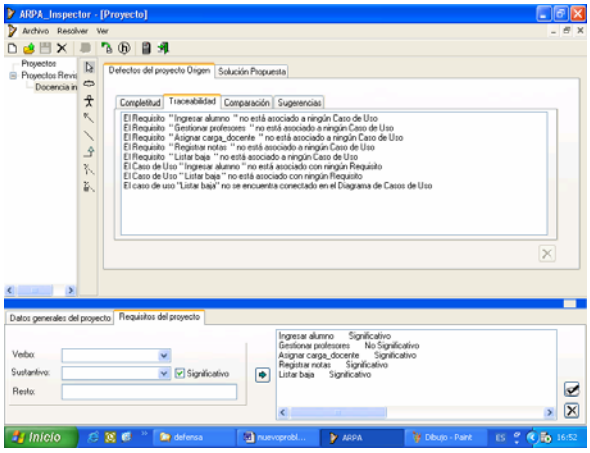
1. Ingresar alumno. (significativo)
2. Gestionar profesores.
3. Asignar carga. (significativo)
4. Registrar nota. (significativo)
5. Listar Bajas. (significativo)

Diagrama de Casos de Uso del Sistema:



Muestra de defectos generados para una nueva revisión





4. Conclusiones

En el trabajo se ha presentado un paquete de herramientas para la gestión de revisiones basado en casos que da soporte al modelo MGRSoft, pero que pueden ser extendidas a otros modelos y que permiten hacer un análisis de la gestión de revisiones, así como la toma de medidas oportunas durante la propia ejecución del proceso de desarrollo de software.

Contar con un conjunto de herramientas que permitan dar seguimiento a la gestión de las revisiones en la empresa, a partir de los datos recogidos como resultado de los proyectos revisados, es una alternativa que dotará a la empresa de la información necesaria para el mejoramiento continuo del proceso.

Además la integración dentro de RevisionCASE de la herramienta ARPA, asistente que utiliza mecanismos de RBC permite acumular la experiencia en las revisiones y emplearla en la detección de defectos en proyectos con características similares.

5. Bibliografía

[1] Althoff K., “Case-Based Reasoning. Handbook of Software Engineering and Knowledge Engineering.”, kaiserslautern, Alemania. Fraunhofer Institute for Experimental Software Engineering (IESE), 2001.

[2] August, S., “Ending Requirements Chaos”, IBM Rational, IBM Corporation 2004, Julio 2004, http://www-136.ibm.com/developerworks/rational/library/4853.html.

[3] Basili, V. y otros, "Software Defect Reduction Top 10 List" Computer, No 34, vol. 1, pág. 135-137, January 2001.

[4] Biffl S., “Analysis of the impact of reading technique and inspector capability on individual inspection performance”, Seventh Asia-Pacific Software Engineering Conference (APSEC'00), IEEE Inc., Diciembre 2000.

[5] Institute of Electrical and Electronics Engineers, “Standard for Software Reviews” IEEE Std. 1028-1997.

[6] Institute of Electrical and Electronics Engineers, “Standard for Software Verification and Validation” IEEE Std. 1012-1998.

[7] Institute of Electrical and Electronics Engineers, “Guide to the Software Engineering Body of Knowledge: trial version”, A proyect of the Software Engineering Coordinationg Committee, The Institute of Electrical and Electronics Engineers Inc, 2001.

[8] Internacional Organization for Standarization, “ISO/IEC 14598-3: 2000. Software Engineering – Product Evaluation. Part 3: Process for developers”, 2000.

[9] Internacional Organization for Standarization, “ISO 9000:2000. Sistemas de Gestión de Calidad-Fundamentos y Vocabulario”, 2000.

[10] Internacional Organization for Standarization, “ISO 9001:2000. Sistemas de Gestión de Calidad-Requisitos”, 2000.

[11] Internacional Organization for Standarization, “ISO 19011:2002. Directrices para la auditoria de los sistemas de gestión de la calidad y/o ambiental.”, traducción certificada, 2002.

[12] Laguna, M.; García, F.; López, O., "Reutilización de Requisitos de usuarios: El modelo Mecano", Proyecto "DOLMEN" de la CICYT TIC2000-1673-C06-05, Ministerio de Ciencia y Tecnología. España.

[13] Manjares A., “Razonamiento basado en casos.” Universidad Nacional de Educación a Distancia, Departamento de Inteligencia Artificial, Madrid, España, 2001.

[14] McEwen, S., “Requirements: An introduction”, IBM Rational, IBM Corporation 2004, Abril 2004, http://www-136.ibm.com/developerworks/rational/library/4166.html.

[15] O'Neill, D., "National Software Quality Experiment: Results 1992-1999." Software Technology Conference, Salt Lake City, 2000.

[16] Software Engineering Institute, “Capability Maturity Model Integration (CMMISM), Version 1.1”, CMU/SEI-2002-TR-011, ESC-TR-2002-011, CMMI Product Team, Pittsburgh PA SEI, Carnegie Mellon University, March 2002.

Modelo Liviano de Calidad para la Mejora de Procesos de Desarrollo de Software

Carmen J. Sánchez[1], Maria E. Solís[1], Francisco J. Pino[1,2], Julio A. Hurtado[1,3]
[1]*Grupo IDIS, Universidad del Cauca, Colombia*
[2]*Grupo Alarcos, Universidad Castilla – La Mancha, España*
[3]*Departamento de Ciencias de la Computación, Universidad de Chile, Chile*
{cjsanchez, msolis, fjpino, ahurtado}@unicauca.edu.co

Resumen

En la comunidad de Ingeniería del Software existe una tendencia generalizada de plantear estrategias que permitan elevar el nivel de competitividad de las empresas de software. Dada la dificultad de implantar modelos de calidad de estándares internacionales en las micro, pequeñas y medianas empresas (MiPyMEs) desarrolladoras de software, actualmente una estrategia adecuada es la definición de modelos adaptadores que faciliten la adopción e implantación de los estándares de organizaciones internacionales como el SEI e ISO, entre otros. Este artículo presenta un modelo liviano de calidad para la mejora de procesos de desarrollo software, denominado MLCal-PDS, ajustado a las necesidades de la industria del software colombiana compuesta en su gran parte por MiPyMES. El modelo adopta las mejores prácticas de los modelos de calidad más reconocidos internacionalmente, además integra algunas de las características del manifiesto ágil y define dos componentes fundamentales: un modelo liviano de referencia (MLRef-PDS) y un modelo liviano de evaluación (MLEva-PDS). El modelo es una guía con la cual se intenta orientar a las empresas desarrolladoras de software para que mejoren la calidad en sus procesos y además pretende que su aplicación en una MiPyME sea fácil de implantar y gestionar.

1. Introducción

La industria de software de Colombia está compuesta principalmente por micro, pequeñas y medianas empresas – MiPyMES [18], estas empresas de software tienen serios problemas de madurez en sus procesos de desarrollo y en la mayoría de los casos los procesos son caóticos en su operación y afectan a toda la empresa [9]. Las empresas planean asegurar la calidad de sus productos a través de: la mejora del proceso y la acreditación en modelos de calidad del SEI ó ISO [6]. Sin embargo, la preparación previa a la acreditación es larga y costosa, porque los modelos de mejora, proceso y evaluación de las organizaciones como el SEI e ISO están estructurados (y han sido construidos) para ser aplicables a grandes empresas, por lo que la aplicación de éstos es difícil en las MiPyMES debido a que un proyecto de mejora supone una gran inversión en dinero, tiempo y recursos, además las recomendaciones son complejas de aplicar y el retorno de la inversión se produce a largo plazo [9, 11, 16, 26].

Así pues dada la dificultad de implantar modelos de calidad de estándares internacionales (tales como CMMI [3], SCAMPI[2], ISO/IEC 12207[4], ISO/IEC 15504:2004 [5], IDEAL [20]) en las MiPyMEs desarrolladoras de software, actualmente una estrategia adecuada es la definición de modelos adaptadores que faciliten la adopción e implantación de éstos estándares creados por organizaciones internacionales como el SEI e ISO. En el presente articulo se presenta un "Modelo Liviano de Calidad para la Mejora de Procesos de Desarrollo Software" (denominado MLCal-PDS) el cual es un modelo que esta cimentado en algunos de los estándares de mejora de procesos más reconocidos internacionalmente como son CMMI, ISO/IEC 12207 y el ISO/IEC 15504. El modelo define dos componentes fundamentales: un modelo liviano de referencia (MLRef-PDS) y un modelo liviano de evaluación (MLEva-PDS), además es una guía con la cual se intenta orientar a las empresas desarrolladoras de software para que mejoren la calidad en sus procesos. El modelo pretende que su aplicación en una MiPyME sea de fácil implantación y gestión.

Para la construcción de éste modelo se ha analizado y sintetizado los componentes de procesos descritos en los estándares nombrados anteriormente, luego se han factorizado y adaptado algunos elementos (mejores

prácticas) para luego ajustarlos a las características propias de la industria del software colombiana. Estas características se han extraído mediante una investigación realizada en algunas empresas desarrolladoras de software del sur occidente colombiano [18]. Además el modelo recopila algunas características del manifiesto ágil [10] para que el modelo sea liviano y este acorde a las características del contexto socio-económico del país y pueda ser aplicado a su industria de software, sirviendo como guía para que las empresas del sector aseguren calidad en su proceso de desarrollo a través de la mejora de procesos software.

Además de la presente introducción el artículo presenta en la sección 2 el estado del arte y trabajos relacionados. En la sección 3 se muestra la definición del modelo liviano de calidad MLCal-PDS. La sección 4 muestra la estructura y características del modelo y la sección 5 muestra las conclusiones y futuros trabajos.

2. Estado del Arte y Trabajos Relacionados

En el mundo existe una preocupación por la calidad de los procesos de desarrollo de software como elemento para incrementar la competitividad de las diferentes industrias nacionales, actualmente una estrategia muy utilizada para este propósito es la mejora de los procesos software - SPI (Software Process Improvement). SPI involucra diferentes tipos de modelos, conocidos comúnmente como modelos de calidad de procesos software, entre los cuales están un modelo que conduce la mejora (por ejemplo IDEAL ó ISO/IEC 15504-4), un método de evaluación de procesos (por ejemplo ISO/IEC 15504-2 ó SCAMPI) y un modelo de procesos de referencia a seguir (por ejemplo CMMI ó ISO/IEC 12207). Además es importante tener en cuenta que la familia de normas ISO 9001:2000 [1] también se ha utilizado en el ámbito de SPI. Hoy por hoy los modelos de calidad más aceptados por la industria del software a nivel mundial son: CMMI, ISO 9001:2000 e ISO/IEC 15504:2004.

Sin embargo estos modelos y estándares normalmente están sobredimensionados al momento de ser aplicados en contextos más pequeños, lo cual ha conducido a iniciativas enfocadas hacia su aplicación PyMES.

La Unión Europea ha impulsado iniciativas como ESSI (European Software and System Initiative) que han promovido diferentes proyectos para fortalecer SPI en Pymes_DS, como por ejemplo SPIRE (Software Process Improvement in Regions of Europe)[1], TOPS (Toward Organised Software Processes in SMEs) [3] ó PROCESSUS [17], entre otros.

En Europa se han realizado investigaciones en donde se realizan ajustes de modelos de mejora como IDEAL para ser aplicados a PyMES, el estudio presentado en [13] es un buen ejemplo de esto. También MESOPYME [12] ha sido creado en España para guiar la mejora en pequeñas empresas y en Australia el proyecto IMPACT [28] también persigue este objetivo.

En el caso de México, se ha desarrollado el modelo MoProSoft [21] - Modelo de Procesos para la Industria de Software en México, el cual tiene como propósito fomentar la estandarización de su operación a través de la incorporación de las mejores prácticas en gestión e ingeniería de software.

En el caso de Brasil, se ha desarrollando el proyecto mps Br [29], cuyo objetivo principal es definir e implementar un modelo para la mejora de procesos de software orientado a las micro, pequeñas y medianas empresas de forma que estas obtengan un nivel de madurez 2 o 3 de CMMI a un costo accesible. El proyecto propone dos modelos: un Modelo de Referencia para la mejora del proceso del software – MR mps y un Modelo de Negocio para la mejora del proceso del software – MN mps. El Modelo de Negocio para la mejora del proceso del software, define los elementos e interacciones involucrados para la certificación de la empresa a través de la implementación de MR de dos maneras: personalizada para una empresa o conjunta entre un grupo de empresas, logrando así costos más accesibles para las empresas micro, pequeñas ó medianas.

El proyecto SIMEP-SW desarrollado en Colombia cuyo principal resultado es el marco de trabajo Agile SPI [19], basa su estrategia de mejora en proporcionar a la organización un proceso ágil que guía un programa SPI, el cual cuenta con los elementos básicos para hacer posible que una MiPyME pueda adelantar esfuerzos de mejora acorde a sus necesidades.

Una iniciativa integradora de las diferentes propuestas creadas para Iberoamérica nace con el proyecto CompetiSoft [8] cuyo objetivo es el de incrementar el nivel de competitividad de las PyMES Iberoamericanas productoras de software mediante la creación y difusión de un marco metodológico común que, ajustado a sus necesidades específicas, pueda llegar a ser la base sobre la cual establecer un mecanismo de evaluación y certificación de la industria del software reconocido en toda Iberoamérica.

El “Modelo Liviano de Calidad para la Mejora de Procesos de Desarrollo Software - MLCal-PDS” ha sido desarrollado en el marco de los proyectos SIMEP-SW y Competisoft. MLCal-PDS se diferencia con respecto a otros modelos propuestos en estos aspectos:

- Esta integrado a un framework de mejora ágil, en cuyo contexto aprovecha las metodologías y

herramientas (conceptuales y software) para su aplicación y gestión.

- Es un modelo de calidad adaptado a las necesidades actuales de la industria del software del suroccidente colombiano, configurado a este contexto y por tanto tiende a ser un modelo prescriptivo más que descriptivo.
- Introduce tres elementos claves: (i) Gestión de conocimiento como disciplina de proceso; (ii) Agilidad como atributo de medida para determinar cuando la disciplina vuelve inmanejable un proceso en una MiPyME y (iii) Innovación como atributo para no descargar toda la responsabilidad de madurez en el proceso, sino ayudarle con elementos innovadores que simplifiquen o automaticen tareas en el desarrollo de software.

3. Visión general del Modelo Liviano de Calidad para la Mejora de Procesos de Desarrollo Software

El resultado del proyecto SIMEP-SW es Agile SPI, que sigue la premisa que los modelos desarrollados sean ligeros y basados en estándares internacionales, acordes a las características, idiosincrasia y realidad socio- económica de la naciente industria del software en el sur occidente Colombiano. La arquitectura de Agile SPI, se presenta en la siguiente figura.

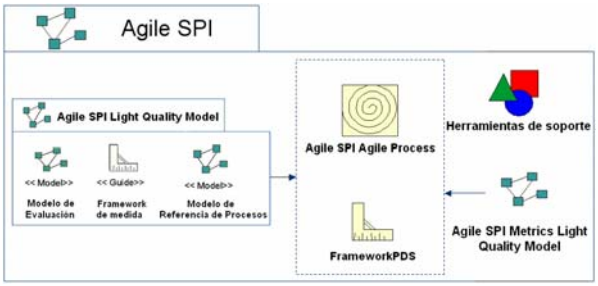


Figura 1. Componentes de Agile SPI

El "Modelo Liviano de Calidad para la Mejora de Procesos de Desarrollo Software – MLCal-PDS" es una concreción del componente Agile SPI Light Quality Model. Algunas características generales son:

- Diferencia entre el modelo de referencia de procesos y el modelo de evaluación para facilitar su comprensión y utilización, proporcionando a las empresas facilidad para orientarse adecuadamente hacia un modelo u otro según la necesidad que requiera implementar.
- Conserva una estructura sencilla buscando una adecuada comprensión de las áreas de procesos que lo componen y adiciona a esta estructura algunos elementos para facilitar su utilización, tratando de guiar la forma adecuada para la implementación de los procesos en la organización.
- Mantiene un conjunto de áreas de proceso para facilitar la implementación y evaluación de procesos individuales. No los liga a ningún nivel de madurez, intentando proporcionar de esta manera flexibilidad a las empresas para elegir los procesos en los que quiere establecer la mejora.
- Proporciona un marco definido de evaluación, adquiriendo los elementos necesarios del modelo de referencia para la evaluación, adicionando medidas para el rendimiento y gestión de los procesos y dando la oportunidad de conocer la agilidad e innovación de los procesos que la MiPyME lleva a cabo.

MLCal-PDS esta compuesto por dos modelos: un modelo liviano de referencia de procesos (MLRef-PDS) y un modelo liviano de evaluación (MLEva-PDS) para tener de una forma más organizada y separada la guía de referencia de implementación de procesos de la guía de evaluación de procesos.

El modelo liviano de referencia de procesos (MLRef-PDS), presenta las mejores prácticas (extraídas de modelos internacionales) para el desarrollo de software. Se organizan en áreas de proceso y se definen los roles necesarios en cada área y los productos de trabajo que se deben obtener en la implementación de estas prácticas.

MLRef-PDS es una guía que le permite a la empresa conocer los procesos que debe adoptar para iniciar una cultura de calidad basada en las prácticas iniciales que se deben adoptar para un determinado proceso o para los procesos que tenga una organización. Teniendo en cuenta que muchas MiPyMEs se encuentran en estados caóticos y no tienen ningún proceso definido, el modelo de referencia aporta una serie de procesos que permiten que la empresa los instaure, como un primer paso para empezar a mejorar sus procesos. Esto permite que se comiencen a desarrollar una serie de actividades y un número de productos de trabajo que garantizan que el proceso se está llevando a cabo de una forma adecuada y controlada. De esta manera la MiPyME comienza a adoptar características de los modelos de calidad internacionales, lo cual le permite escalar a un nivel más alto en la capacidad de sus procesos y a futuro buscar una certificación.

El modelo liviano de evaluación (MLEva-PDS), permite valorar los procesos de una organización con respecto al modelo de referencia de procesos. Ofrece una serie de medidas que contribuyen a evaluar el rendimiento de un proceso teniendo en cuenta las características del modelo de referencia, basándose en

dos indicadores: (i) subprácticas realizadas en cada proceso y (ii) productos de trabajo obtenidos en el proceso. Estos indicadores se toman del modelo de referencia de procesos, siguiendo la estructura presentada en la norma ISO/15504-5:2006 [7].

MLEva-PDS se basa en Light MECPDS [23, 24] y adicionalmente presenta dos atributos de calidad:

- Uno para valorar la agilidad con que se desarrollan los procesos teniendo en cuenta los principios y valores del manifiesto ágil. Este atributo intenta verificar que los procesos que se implantan en una MiPyME sean ágiles pero con responsabilidad.
- El otro para valorar la innovación que intenta verificar el grado de apalancamiento tecnológico que la empresa introducir para mejorar sus procesos.

Mediante la realización de este modelo se busca motivar a las empresas a que inicien una cultura hacia el mejoramiento continuo, implantando en primera instancia el MLCal-PDS, ya que permite establecer procesos que son primordiales para un buen funcionamiento y control de procesos y productos; buscando como primera medida organizar la empresa mediante la definición de estos procesos realizando las subprácticas y productos de trabajo que el modelo de referencia sugiere y posteriormente con el modelo de evaluación corroborar la mejora de una empresa.

4. Componentes del MLCal-PDS

El interés del MLCal-PDS se centra en proponer un conjunto de procesos de desarrollo de software que consideramos son mas prioritarias para las MiPyMEs, que les pueda permitir visualizar y establecer sus propios procesos de desarrollo de software, alineados con la mejores prácticas propuestas por estándares internacionales. El "Modelo Liviano de Calidad para la Mejora de Procesos de Desarrollo Software" presenta la siguiente estructura (ver figura 2):

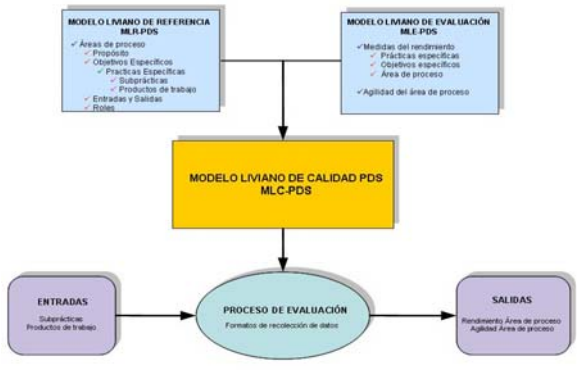


Figura 2. Estructura del MLCal-PDS

Antes de presentar los componentes fundamentales, se presenta una visión general de los elementos utilizados para seleccionar los procesos del modelo.

4.1. Escogencia de las áreas de proceso del MLCal-PDS.

Para la priorización, selección y definición de los procesos que contiene el modelo liviano de referencia se tuvo en cuenta varios elementos:

- Una contrastaron las mejores prácticas presentadas en los estándares CMMI e ISO/IEC 12207.
- Una revisión sistemática acerca de los esfuerzos de mejora de procesos software (SPI) llevados a cabo en PYMES desarrolladoras de software [22] cuyo objetivo es conocer lo que se ha realizado y logrado sobre SPI en este tipo de empresas.
- Una investigación sobre el estado de la práctica de los procesos de desarrollo de software en la región del suroccidente de Colombia [18].
- Revisión de la literatura de trabajos relacionados con SPI en MiPyMEs.

La investigación sobre el estado de la práctica de los procesos de desarrollo de software en la región del suroccidente de Colombia se realizó en el proyecto SIMEP-SW. La investigación involucró una muestra de 20 empresas ubicadas en las ciudades de Cali, Popayán y Pasto, algunas de las consideraciones sobre las MiPyMEs, se presentan a continuación:

- El perfil general de las empresas es que un 70% son micro y un 30% son pequeñas, con una trayectoria en su mayoría de menos de 5 años, y una minoría de 10 o más años; con un alcance o proyección de mercado casi en su totalidad del ámbito nacional.
- Se caracterizan además por adoptar metodologías de desarrollo diferentes dependiendo de la naturaleza del proyecto que realice. Algunas manejan un proceso de desarrollo propio.
- Existe una gran falencia con respecto al proceso de desarrollo que se maneja, la mayoría de las empresas no tienen la capacidad de adaptar procesos de desarrollo a su entorno, sino que prefieren adaptarse a los procesos existentes.
- Presentan el "síndrome de la programación extrema (XP)", es decir a un proceso caótico, sin una definición clara de entradas y salidas, responsables, recursos, centrado en la generación de código, entre otros, se le denomina XP. Hay un desconocimiento claro que la programación extrema no es solo generar código fuente, sino que involucra una serie de compromisos, que la

organización debe asumir, para garantizar que el proceso sea ágil y no "frágil".

Entre otras cosas, la investigación recopiló a través de una encuesta, a las empresas involucradas, información sobre las técnicas y prácticas que se utilizan en algunas disciplinas o áreas de proceso para el desarrollo de software. Las disciplinas consideradas en la investigación fueron: Modelado de negocios, Ingeniería de requisitos, Análisis y diseño, Implementación, Pruebas, Implantación, Planificación, Seguimiento, Administración de requisitos, Gestión de la configuración y Aseguramiento de calidad. Para cada disciplina se indagó por una serie de artefactos, técnicas y prácticas para determinar cuales son utilizadas por las MiPyMEs. Con esta información recolectada se generó un perfil sobre el estado de las prácticas para el desarrollo del software de las MiPyMEs del sur occidente Colombiano.

Realizando un análisis de la información recogida sobre los artefactos, técnicas y prácticas utilizadas en cada disciplina se determinó el grado de implementación promedio de cada de éstas (ver figura 3). En esta figura se ilustra la cantidad de artefactos utilizados para la implementación de cada disciplina.

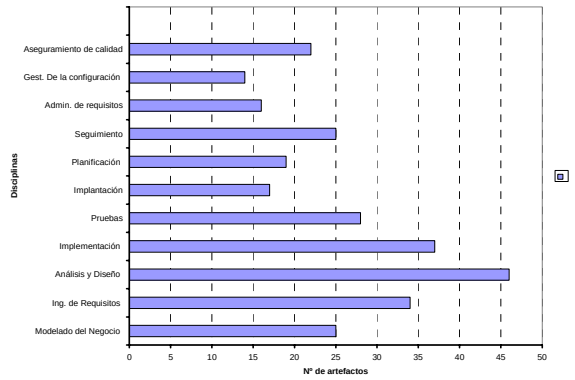


Figura 3. No. Artefactos por cada disciplina

Además, se aprecia en la "Revisión sistemática de mejora de procesos software en micro, pequeñas y medianas empresas" presentada en [22] que los esfuerzos de mejora apuntan a mejorar procesos como la gestión del proyecto, documentación, gestión de cambio de requisitos, establecimiento de procesos, gestión de la configuración y obtención de requisitos. Hay otros procesos como por ejemplo el de revisión conjunta, adquisición, suministro, entre otros, sobre los cuales no se reporta ningún tipo de mejora.

Después del estudio, análisis y síntesis de todos trabajos tomados en cuenta, las disciplinas consideradas prioritarias por MLCal-PDS en relación con las prácticas a establecer primero cuando inicien un programa SPI son:

- Gestión de proyectos
- Administración de requisitos
- Desarrollo de requisitos
- Solución técnica
- Gestión de la configuración
- Validación y Verificación
- Medición y análisis
- Aseguramiento de calidad
- Gestión del conocimiento.

La medición y análisis se ha propuesto porque es una responsabilidad clave en la gestión y mejora de procesos software. Las medidas son la base para detectar desviaciones del funcionamiento aceptable del proceso, además son también la base para identificar las oportunidades para la mejora de proceso [15].

La gestión de conocimiento surge de la tesis 6 de Conradi-Fugetta [14] donde expresa que la mejora del proceso software es aprendizaje no control y de la connotación de industria de conocimiento de la misma industria de software. Este modelo encapsula todo lo referente al aprendizaje en la disciplina de gestión del conocimiento.

4.2. Modelo Liviano de Referencia (MLRef-PDS)

Como se indicó anteriormente el modelo liviano de referencia involucra las mejores prácticas para el proceso de desarrollo software, logradas a partir de la comparación y contrastación de CMMI y la norma ISO/IEC 12207. Esta comparación y contrastación se desarrolló después de realizar un estudio minucioso y disciplinado de los modelos seleccionados como base para el desarrollo del modelo de referencia.

Debido a que la estructura de CMMI es diferente a la norma ISO/IEC 12207 y a que organizan las prácticas de desarrollo de software de diferente forma, no se pudo desarrollar una comparación uno a uno. Sin embargo las correspondencias que predominaron fueron entre áreas de proceso y prácticas específicas de CMMI con procesos y tareas de ISO/IEC 12207.

4.2.1. Componentes del MLRef-PDS. Este modelo esta compuesto por Áreas de proceso y estas a su vez contienen los elementos más importantes de los modelos de referencia estudiados. El patrón de procesos trata de disminuir la complejidad facilitando su manejo; como se muestra a continuación:

- *Nombre del área de proceso*: Nombre que se da a un conjunto de prácticas relacionadas, que cuando se desarrollan colectivamente, satisfacen un objetivo específico que asegura el establecimiento de la mejora.

- *Propósito*: Objetivos generales medibles y resultados esperados de la implantación efectiva el área de proceso.
- *Descripción*: Descripción general del área de proceso y de las actividades que componen el flujo de trabajo del área de proceso.
- *Rol responsable y ejecutor*: Roles involucrados en la implementación del área de proceso. El rol responsable es el rol principal de validar y vigilar la ejecución de las prácticas involucradas en el área de proceso; y el rol ejecutor es el responsable de la ejecución de las actividades de las áreas de proceso.
- *Objetivos Específicos*: Objetivo cuya finalidad es asegurar el cumplimento del propósito del área de proceso.
- *Prácticas Específicas*: Prácticas de las cuales esta compuesto el objetivo especifico, y que tienen como finalidad asegurar el cumplimiento del objetivo al que pertenecen.
- *Productos de trabajo*: Productos generados por la práctica específica a la que se encuentran asociados. Estos productos de trabajo formarán parte de las entradas y salidas del área de proceso a la que pertenecen dichas prácticas específicas.
- *Subprácticas*: Actividades o tareas asociadas a las prácticas específicas. Tienen como finalidad asegurar el cumplimento de la práctica específica a la que se encuentran asociadas. Tendrán también cada una de ellas, productos de trabajo asociados (entradas y/o salidas) con el fin de establecer mayor claridad para la implementación del área de proceso.

Los productos de trabajo, objetivos específicos y prácticas específicas mantienen un identificador para su posterior utilización en el modelo de evaluación.

A continuación se presenta un ejemplo de un área de procesos definida por el modelo (por efectos de espacio solo se muestra un resumen).

- *Nombre del área de proceso*: Aseguramiento de calidad
- *Propósito*: El propósito del aseguramiento de calidad del proceso y del producto es el de proveer y administrar con una visión objetiva los procesos asociados a los productos de trabajo proporcionando así la seguridad apropiada de que los productos y procesos software del ciclo de vida del proyecto son conformes a los requisitos especificados y se adhieren a los planes del proyecto.
- *Descripción*: El aseguramiento de la calidad del proceso y del producto permite la entrega de productos de alta calidad suministrando al personal y a los encargados del proyecto una apropiada visibilidad y una retroalimentación de los procesos y productos asociados al trabajo a través del ciclo de vida del proyecto. Este proceso implica lo siguiente: (i) Evaluar objetivamente los procesos realizados, los productos de trabajo contra las descripciones, estándares, y procedimientos. (ii) Identificar y documentar los problemas no cumplidos. (iii) Proporcionar una retroalimentación al personal y a los encargados del proyecto en cuanto a los resultados de las actividades de aseguramiento de calidad. (iv) Asegurar que los problemas no resueltos sean manejados. (v) El aseguramiento de calidad puede hacer uso del resultado de otros procesos, tales como verificación y validación.
- *Rol Responsable y ejecutor*: Grupo de aseguramiento de calidad: participa en la revisión de los productos seleccionados para determinar si son conformes o no a los procedimientos, estándares o criterios especificados, siendo totalmente independiente del equipo de desarrollo. Sus funciones están dirigidas a: (i) Identificar las posibles desviaciones en los estándares aplicados, así como en los requisitos y procedimientos especificados. (ii) Comprobar que se han llevado a cabo las medidas preventivas o correctoras necesarias.
- *Objetivos Específicos y prácticas específicas*
  OE1. Evaluar los procesos y productos de trabajo objetivamente.
  PE1. Establecer un proceso de aseguramiento de calidad.
  PE2. Evaluar procesos y productos de trabajo objetivamente.
  OE2. Identificar y registrar problemas.
  PE1. Comunicar problemas encontrados y asegurarse que sean resueltos.
  PE2 Establecer registros.
- *Productos de trabajo:* Plan de aseguramiento de calidad. Objetivos de calidad. Reportes de evaluación. Reportes de problemas no resueltos. Reportes de acciones correctivas. Registros de trazabilidad. Reportes de aseguramiento de calidad. Criterios de evaluación. Lista de los productos de trabajo para evaluación. Reportes de análisis. Registros de comunicación.
- *Subprácticas:* Actividades o tareas asociadas a las prácticas específicas del área de proceso de aseguramiento de calidad.

4.3. Modelo Liviano de Evaluación (MLEva-PDS)

Hay que tener en cuenta que además del modelo de referencia de procesos, otro elemento importante de un programa SPI es el modelo de evaluación de procesos. La evaluación de los procesos software tiene como objetivo detectar aspectos de un proceso software que se pueden mejorar. Para esto es preciso contar con un modelo de evaluación ya que, para poder comenzar la mejora de los procesos software, es muy importante establecer previamente un marco de evaluación para conocer los puntos fuertes y débiles de los procesos de una organización. Es necesario entonces la definición del modelo liviano de evaluación, el cual considere una forma ágil y confiable de evaluar procesos.

4.3.1. Estructura del MLEva-PDS. El modelo liviano de evaluación MLEva-PDS se basa en Light MECPDS [23, 24]. Light MECPDS es una concreción y adaptación (del inglés tailoring) de la norma ISO/IEC 15504 [5, 7] para la valoración de procesos en MiPyMEs. Los propósitos de Light MECPDS, son:

- Establecer los elementos necesarios para evaluar el cumplimiento y capacidad de los procesos de una organización, con respecto a un modelo de referencia de procesos.
- Fomentar la evaluación en las MiPyMEs de desarrollo de software del sur occidente Colombiano, con el objetivo de conocer sus puntos fuertes y débiles, para que sirvan de guía en la mejora de los procesos de desarrollo de software de la organización.
- Aportar un modelo de evaluación ligero para que sea aplicable a las MiPyMEs, de manera fácil y económica, con pocos recursos y en poco tiempo.

Para aligerar el modelo de evaluación, Light MECPDS describe la evaluación con respecto al nivel dos de madurez de ISO/IEC 15504, y su correspondiente framework de medida.

El framework de medida es tomado de la norma ISO/IEC 15504 [5] y especifica varios niveles de capacidad definidos por una escala jerárquica de tres niveles: (i) Nivel 0. Proceso Incompleto, (ii) Nivel 1. Proceso Realizado y (iii) Nivel 2. Proceso Gestionado. El alcanzar un nivel se demuestra por el cumplimiento de atributos de proceso, que son elementos que permiten determinar las capacidades y habilidades de un proceso y están compuestos por prácticas de gestión. Los atributos de proceso definidos son: PA 1.1 Realización del proceso, PA 2.1 Gestión de la realización y PA 2.2. Gestión del producto de trabajo.

El framework de medida de Light MECPDS se define para dos dimensiones: *capacidad del proceso* y *cumplimiento del proceso*.

En la *dimensión del cumplimiento del proceso* Light MECPDS utiliza el mapeo de los propósitos y descripción además de los objetivos y prácticas especificas seleccionados del modelo de referencia de procesos (MLRef-PDS) como indicadores de evaluación. En ésta dimensión se evalúa el atributo de proceso PA 1.1 Realización del proceso, que pertenece al Nivel de capacidad 1: Proceso Realizado.

En la *dimensión de la capacidad del proceso* utiliza el mapeo de los atributos de calidad PA 2.1 Gestión de la realización y PA 2.2. Gestión del producto de trabajo, del framework de medida, como indicadores de evaluación. Esta dimensión se evalúa por los atributos de proceso descritos anteriormente, los cuales pertenecen al Nivel de capacidad 2: Procesos Gestionado.

El modelo liviano de evaluación MLEva-PDS adicionalmente propone dos atributos de calidad, que no están definidos en la norma, pero que se consideran importantes en el entorno de la evaluación de procesos en un MiPyME, estos son: (i) PA 1.2 Agilidad del proceso, (ii) PA 2.3 Innovación del proceso.

4.3.2. Atributo agilidad del proceso. Éste atributo permite que una pequeña organización desarrolladora de software evalúe el nivel de agilidad de los procesos utilizados por ella, para el desarrollo de sus productos software. Es adicional a los atributos definidos anteriormente y pertenece al Nivel de capacidad 1: Procesos Realizado. Es importante que las PyMES desarrolladoras de software asuman la agilidad con responsabilidad. Es decir que si los procesos ágiles les aportan una ventaja competitiva debido a su estructura interna, estos procesos al menos deben tener un mínimo de elementos que sustenten que sus procesos son ágiles porque siguen los lineamientos del manifiesto ágil. De esta forma se evalúa, además de la realización del proceso, si el proceso se realiza de manera ágil. El atributo de proceso PA 1.2 se presenta en la siguiente tabla.

Tabla 1. Atributo PA 1.2 Agilidad del proceso

Id. Atributo	**Descripción del atributo:** Agilidad del proceso		**Escala**
PA 1.2	El atributo agilidad de proceso es una medida del nivel de agilidad de un proceso.		NI, PI, AI, CI
Nivel	**Id. Practica**	**Descripción de la practica de gestión**	NI, PI, AI, CI
1. Realizado	MP 1.2.1	Tomar acciones para que los responsables tengan la autonomía de organizar el equipo y su comunicación, y las actividades del proceso.	
	MP 1.2.2	Identificar los productos de trabajo que sirvan para tomar decisiones y que aporten valor al cliente en el proceso	
	MP 1.2.3	Definir la colaboración del cliente en el proceso.	
	MP 1.2.4	Definir una estrategia para responder a los cambios de los productos de trabajo que los clientes desean en el proceso.	

Este atributo es aplicable a cualquier área de proceso del modelo liviano de referencia, adquiriendo así las características que tienen los atributos de capacidad de la norma ISO/IEC 15504. Puede ser aplicado por cualquier organización que quiera evaluar la agilidad de sus procesos. Este atributo mide el grado de cumplimiento de algunos de los principios del manifiesto ágil (tres principios no encajaron de manera natural en la definición del atributo de calidad y por ello quedan fuera de la tabla anterior).

4.3.3. Atributo de innovación del proceso. Éste atributo permite que una pequeña organización desarrolladora de software evalúe el nivel de innovación de los procesos utilizados por ella para el desarrollo de sus productos software. Es adicional a los atributos definidos anteriormente y pertenece al Nivel de capacidad 2: Procesos Gestionado. La mejora desde la perspectiva de la innovación de los procesos tiene un retorno a la inversión mucho rápido (si esta es llevado cabo exitosamente) que la mejora desde la perspectiva de gestión de la organización (sin embargo su desventaja es que es mas costoso). Se pretende con este atributo que las MiPyMEs tomen consciencia de la importancia que el apalancamiento tecnológico tiene para mejorar sus procesos software y por tanto la calidad de sus productos. El atributo de proceso PA 2.3 se presenta en la siguiente tabla.

Tabla 2. Atributo PA 2.3 Innovación del proceso

Id. Atributo	**Descripción del atributo:** Innovación del proceso		**Escala**
PA 2.3	El atributo innovación de proceso es una medida del nivel de innovación de un proceso.		NI, PI, AI, CI
Nivel	**Id. Practica**	**Descripción de la practica de gestión**	NI, PI, AI, CI
2. Gestionado	MP 2.3.1	Tomar acciones para que los responsables tengan el liderazgo de organizar actividades de I+D asociadas a las componentes del proceso.	
	MP 2.3.2	Velar porque los componentes del proceso sean diseñados de manera efectiva a partir de enfoques novedosos provistos por la investigación académica e industrial accesible a la organización.	
	MP 2.3.3	Identificar a través de medidas si la introducción de un enfoque novedoso impacta al proceso en términos de simplicidad, tiempo y costo.	
	MP 2.3.4	Introducir enfoques novedosos y tradicionales que son apalancados por herramientas de soporte que permiten minimizar trabajo repetitivo del proceso.	

Este atributo de proceso es aplicable a cualquier área de proceso del modelo liviano de referencia, adquiriendo así las características que tienen los atributos de capacidad de la norma ISO/IEC 15504. Puede ser aplicado por cualquier organización que quiera evaluar la innovación de sus procesos. Este atributo mide el grado de apalancamiento tecnológico del área (la adopción de estrategias conceptuales, técnicas de reuso de software, de aprendizaje, de trabajo colaborativo, de gestión, etc.) de tal forma que la empresa pueda introducir mejoras que se vean reflejadas en la simplificación de procesos, métodos y técnicas, para que las disciplinas sean mas simples de aplicar y de gestionar.

4.3.4. Definición de niveles de capacidad. El framework de medida especifica una escala jerárquica de tres niveles para definir los niveles de capacidad: Proceso Incompleto (nivel 0), Proceso Realizado (nivel 1) y Proceso Gestionado (nivel 2).

La tabla 3 define el nivel de capacidad asociado a un proceso, el cual permite medir el grado de calidad de un producto de software generado por el mismo. Hay una relación entre niveles de capacidad y el grado de cumplimiento de los atributos de proceso evaluados.

Tabla 3. Cumplimiento de niveles de capacidad

Nivel de Capacidad	**Atributos del proceso**	**Grado de cumplimiento esperado**
Nivel 1. Realizado	Realización del proceso	AI o CI
	Agilidad del proceso	AI o CI
Nivel 2. Gestionado	Realización del proceso	CI
	Agilidad del proceso	CI
	Gestión de la realización	AI o CI
	Gestión de los productos	AI o CI
	Innovación del proceso	AI ó CI

4.3.5. Definición de medidas del área de proceso. Las métricas o medidas, se pueden ver como un soporte efectivo a la mejora de procesos, debido a que son esenciales para entender, definir, gestionar y controlar los procesos de desarrollo. Por lo tanto, con el fin de obtener una base cuantitativa para la mejora de procesos software es necesario medir, basándose en los elementos expresados en un modelo de referencia de procesos. Considerando esta necesidad, se han definido un conjunto de medidas, basadas en el trabajo presentado en [25], en donde se expone una metodología para la definición de métricas de rendimiento y capacidad de procesos conforme a la norma ISO/IEC 15504.

Las métricas a nivel del rendimiento del área de procesos han sido definidas con el objetivo de evaluar el grado de cumplimiento de un proceso en relación con un área de proceso definida en MLCal-PDS. Las métricas a nivel de la capacidad del área de procesos han sido definidas con el objetivo de evaluar el grado de capacidad de un proceso en relación con un atributo de proceso definida en MLCal-PDS. Toda la información de las métricas se encuentran en [27]

5. Conclusiones y Trabajo Futuro

Es necesario tener presente el hecho de que en la mayoría de organizaciones existe un factor de resistencia al cambio, por eso el proceso de mejora depende del compromiso de todas las personas de la organización, del convencimiento y la credibilidad que

tengan sobre en el modelo de calidad utilizado con el fin de lograr los objetivos SPI propuestos por la empresa. Consideramos que si el modelo es sencillo pero a la vez robusto, seria de gran ayuda para que sea aceptado e implementado por las personas involucradas en la organización. El modelo presentado en este trabajo pretende ayudar a las MiPyMEs a implantar un programa de mejora que se ajuste a sus necesidades, pero que a la vez sea el primer paso hacia la mejora alineado con modelos internacionales.

El modelo liviano de calidad se construyó bajo un soporte teórico y metodológico estructurado y muy bien definido, basado en modelos y normas internacionales conocidas. El objetivo del modelo es ayudar a definir y evaluar los procesos de desarrollo del software en MiPyMEs cuando inicien un programa de mejora de procesos. SPI tiene como fin optimizar recursos, tiempo y costo en los procesos de la organización para incrementar la satisfacción de sus clientes, generar productos de excelente calidad y logar un perfil de mayor competitividad.

El modelo liviano de calidad proporciona un conjunto de procesos definidos a partir de las mejores prácticas para mantener y desarrollar un mejor software, teniendo en cuenta la realidad de las MiPyMEs del suroccidente Colombiano. Define una forma sencilla de evaluar rendimiento y gestión de los procesos cuando son implantados en una organización. Además propone dos atributos de proceso adicionales a los presentados en la norma ISO/IEC 15504, para evaluar la agilidad y la innovación del proceso.

El modelo liviano de referencia proporciona una estructura práctica que permite disminuir la ambigüedad de los procesos ya que facilita una adecuada comprensión de las áreas de procesos que lo componen y adiciona a esta estructura algunos elementos que a diferencia de otros modelos muestra el ***"cómo"*** se deben desarrollar las prácticas disminuyendo la complejidad de adoptar procesos dejando en claro ***"quiénes"*** son los que contribuyen a la realización de estas prácticas y ***"cuáles"*** son productos que se deben desarrollar para el cumplimiento de un proceso garantizando de esta forma la mejora de procesos en una empresa.

En estos momentos se están documentando cada una de las disciplinas y se están estableciendo los indicadores de medición de las disciplinas de acuerdo al modelo de evaluación. En el contexto de SIMEP-SW se han desarrollado herramientas complementarias que se esperan orquestar en varios proyectos de mejora ya establecidos a través del proyecto y de los cuales también se ha extraído realimentado con información relevante para el desarrollo del modelo. Esta experimentación permitirá validar y ajustar el modelo de calidad a fin de tener una primera versión. La experiencia en el desarrollo y aplicación del modelo, es un insumo valioso para la construcción del modelo iberoamericano propuesto por el proyecto CompetiSoft.

Agradecimientos

Este trabajo ha sido parcialmente financiado por los proyectos: MECENAS (Junta de Comunidades de Castilla-La Mancha, Consejería de Educación y Ciencia, PBI06-0024), COMPETISOFT (Programa Iberoamericano de Ciencia y Tecnología para el Desarrollo – CYTED -, 506PI0287) y ESFINGE (Dirección General de Investigación del Ministerio de Educación y Ciencia, TIN2006-15175-C05-05).

Referencias

[1]. *ISO 9001:2000. Quality management systems - Requirements.* 2000, International Organization for Standardization: Geneva.

[2]. *Standard CMMI® Appraisal Method for Process Improvement (SCAMPI), Version 1.1: Method Definition Document (CMU/SEI-2001-HB-001)*. 2001, Software Engineering Institute (SEI): Pittsburgh.

[3]. *CMMI for Systems Engineering/Software Engineering, Version 1.1*. 2002, Software Engineering Institute (SEI): Pittsburgh.

[4]. *ISO/IEC 12207:2002/FDAM 2. Information technology - Software life cycle processes*. 2004, International Organization for Standardization: Geneva.

[5]. *ISO/IEC 15504-2:2003/Cor.1:2004(E). Information technology - Process assessment - Part 2: Performing an assessment*. 2004, International Organization for Standardization: Geneva.

[6]. *Panorama de la Industria del Software en Latinoamérica*. 2004, Mayer & Bunge Informática LTDA: Brasil. p. 97.

[7]. *ISO/IEC 15504-5:2006(E). Information technology - Process assessment - Part 5: An exemplar Process Assessment Model*. 2006, International Organization for Standardization: Geneva.

[8]. *Mejora de procesos para fomentar la competitividad de la pequeña y mediana industria de software Iberoamericana - COMPETISOFT*. 2006. Available on: http://alarcos.inf-cr.uclm.es/Competisoft. Accessed: November, 2006.

[9]. Batista, J. and A. Figueiredo, *SPI in a very small team: a case with CMM.* Software Process: Improvement and Practice, 2000. Vol. 5(4) December pp. 243-250.

[10]. Beck, K. *Manifesto for Agile Software Development*. 2003. Available. Accessed: October, 2006.

[11]. Calvo-Manzano, J.A., *Métodos de mejora del proceso de desarrollo de sistemas de información en la pequeña y mediana empresa*. 1999, Universidad de Vigo: Vigo.

[12]. Calvo-Manzano, J.A., G. Cuevas, T. San Feliu, A. De Amescua, and M. Pérez, *Experiences in the Application*

of Software Process Improvement in SMES. Software Quality Journal, 2002. Vol. 10(3) November pp. 261-273.
[13]. Casey, V. and I. Richardson, *A practical application of the IDEAL model.* Software Process: Improvement and Practice, 2004. Vol. 9(3) July/September pp. 123-132.
[14]. Conradi, R. and A. Fuggetta, *Improving Software Process Improvement.* IEEE Software, 2002. Vol. 19(4) July/August pp. 92-99.
[15]. Florac, W.A., R.E. Park, and A.D. Carleton, *Practical Software Measurement: Measuring for Process Management and Improvement*. 1997, Pittsburgh, Software Engineering Institute, Carnegie Mellon University pp. 1-12.
[16]. Hareton, L. and Y. Terence, *A process framework for small projects.* Software Process: Improvement and Practice, 2001. Vol. 6(2) Juny pp. 67-83.
[17]. Horvat, R.V., I. Rozman, and J. Györkös, *Managing the complexity of SPI in small companies.* Software Process: Improvement and Practice., 2000. Vol. 5(1) March pp. 45-54.
[18]. Hurtado, J., F. Pino, and J. Vidal, *Estado de la práctica del proceso software en el suroccidente colombiano.* 2006, Universidad del Cauca - Colciencias: Popayán, Colombia.
[19]. Hurtado, J., F. Pino, and J. Vidal, *Software Process Improvement Integral Model: Agile SPI. Technical Report SIMEP-SW-O&A-RT-6-V1.0. 2005.* 2006, Universidad del Cauca - Colciencias.: Popayán, Colombia.
[20]. McFeeley, R., *IDEAL: A Users Guide for Software Process Improvement, Handbook CMU/SEI-96-HB-001*. 1996, Software Engineering Institute, Carnegie Mellon University: Pittsburgh, USA.
[21]. Oktaba, H. *MoProSoft®: A Software Process Model for Small Enterprises.* 2006. Proceedings of the First International Research Workshop for Process Improvement in Small Settings. Pittsburgh, Carnegie Mellon University. pp. 93-101.
[22]. Pino, F., F. Garcia, and M. Piattini, *Revisión sistemática de mejora de procesos software en micro, pequeñas y medianas empresas.* Revista Española de Innovación, Calidad e Ingeniería del Software (REICIS), 2006. Vol. 2(1) Abril pp. 6-23.
[23]. Pino, F., F. Garcia, F. Ruiz, and M. Piattini, *Adaptación de las normas ISO/IEC 12207:2002 e ISO/IEC 15504:2003 para la evaluación de la madurez de procesos software en países en desarrollo.* IEEE Latin America Transactions, 2006. Vol. 4(2) April pp. 16-23.
[24]. Pino, F., F. Garcia, F. Ruiz, and M. Piattini. *A Lightweight Model for the Assessment of Software Processes.* 2006. European Systems & Software Process Improvement and Innovation (EuroSPI 2006). Joensuu, Finland. pp. in press.
[25]. Pino, F., F. Garcia, M. Serrano, and M. Piattini. *Estimating the Performance and Capacity of Software Processes according to ISO/IEC 15504.* 2006. Proceedings of the International Conference on Software Process and Product Masurement - Mensura 2006. Cádiz, Spain. pp. 171-180.
[26]. Saiedian, H. and N. Carr *Characterizing a software process maturity model for small organizations.* ACM SIGICE Bulletin, 1997. Vol. 23(1) July pp. 2-11.
[27]. Sanchez, J. and M. Solis, *Modelo liviano de calidad para la mejora de procesos de desarrollo de software*, in *Departamento de Sistemas*. 2006, Universidad del Cauca: Popayán.
[28]. Scott, L., R. Jeffery, L. Carvalho, J. D'Ambra, and P. Rutherford. *Practical Software Process Improvement -The IMPACT Project.* 2001. Proceedings of the Australian Software Engineering Conference. pp. 182-189.
[29]. Weber, K., E. Araújo, A. Rocha, Machado, D. Scalet, and C. Salviano, *Brazilian Software Process Reference Model and Assessment Method*, in *Computer and Information Sciences*. 2005, Springer Berlin / Heidelberg. p. 402-411.

Diseño y Desarrollo de un Entorno Integrado para Simuladores de Entrenamiento de Procesos Industriales

Pedro A. Corcuera
Dpto. Matemática Aplicada y Ciencias de la Computación
Universidad de Cantabria
Av. de los Castros s/n, 39005 Santander, Spain
corcuerp@unican.es

Resumen

El factor humano es uno de los principales elementos en la seguridad de los procesos industriales, siendo primordial la formación y el entrenamiento de los operadores en todo tipo de situaciones, utilizándose para ello simuladores de entrenamiento. La arquitectura y entornos computacionales que soportan los simuladores corresponden a programas multihilo que son capaces de distribuir y controlar los procesos desde una red de ordenadores. El presente artículo detalla la experiencia en el desarrollo de un simulador de alcance total réplica para una planta nuclear, como extensión de otro simulador desarrollado con paneles virtuales como interfaz de usuario, que utiliza como plataforma de ejecución un ordenador personal bajo sistema operativo Windows. También se ha incorporado la posibilidad de acceder al simulador desde la Intranet con lo que se amplía su uso a todo el personal de operación y la capacidad de integrarlo en cursos tipo e-learning.

1. Introducción

Uno de los factores más importantes para la seguridad de los procesos industriales es el factor humano. En particular en la industria nuclear es primordial la formación y el entrenamiento de los operadores que tienen la responsabilidad de realizar y hacer el seguimiento correcto de los procedimientos de operación normal, anormal y de emergencia. Para ello es necesario que la formación y entrenamiento se realice en un entorno lo más parecido o igual al que desempeñan su trabajo que es la sala de control. Desde esta sala de control el operador dispone de los elementos principales de operación y seguimiento del proceso controlado. Por esta razón, los organismos reguladores de la actividad nuclear de cada país exigen que la formación de los operadores se realice en lo que se denomina simulador réplica de alcance total, es decir un simulador que reproduzca con un alto grado de fidelidad física la sala de control controlado por los programas de ordenador apropiados y que los modelos de la simulación tenga en cuenta los principios de ingeniería básicos y avanzados para reproducir el comportamiento de la planta en una serie de situaciones cubriendo los estado de normalidad y anormalidad que se presentan en el proceso. Este tipo de instalaciones son costosas y complejas tanto en su desarrollo como en su mantenimiento.

Desde el punto de vista de diseño de la interacción con los sistemas de control distribuido utilizados en la mayoría de plantas industriales se pueden distinguir tres tipos de simuladores: Estimulado, Híbrido y Emulado. El simulador estimulado es costoso y requiere de extensas modificaciones para que los sistemas de control puedan atender las operaciones normales de carga de una condición inicial así como para guardarla y no permite detener la simulación. Como ventajas podemos señalar que la interacción hombre máquina es exactamente igual a la planta real y que adicionalmente puede servir para el entrenamiento en los equipos de control. Por su parte, los simuladores totalmente emulados son menos costosos con diferencia y permite al instructor realizar una serie de operaciones como son grabar una condición inicial, volver atrás, parar, avanzar y generar gráficos de tendencias.

Otro tipo de simulador utilizado principalmente para formación o personal de ingeniería es el que se conoce como simulador de escritorio [1], que se ejecuta normalmente en un ordenador personal y que contiene, además del entorno de simulación, todos los elementos de control y seguimiento dispuestos en "paneles virtuales" a través de los cuales el alumno puede interactuar como si estuviera en la sala de control. Este tipo de simulador tiene la ventaja

adicional de ser flexible en su configuración pudiendo ofrecer la opción de modificar la disposición los paneles y modelos.

El presente artículo detalla la experiencia en el desarrollo de un simulador de alcance total réplica para una planta nuclear como extensión de otro simulador de alcance total con paneles virtuales [2] como interfaz de usuario, que utiliza como plataforma de ejecución un ordenador personal. Se describe el entorno de simulación consistente en servidores y clientes multihilo que se comunican utilizando RPC, por lo que es posible distribuir los servicios del simulador como servidor de bases de datos, de los modelos de simulación y la consola del instructor. A continuación se describe las operaciones que ofrece la consola del instructor que están sujetas a normas de los organismos reguladores. También se describe la tecnología de los paneles virtuales basado principalmente en componentes software que permite una gran facilidad y flexibilidad para su diseño y modificación. Una ventaja muy apreciada por los instructores es que los componentes que representan los elementos de control y seguimiento pueden insertarse dentro de la documentación del simulador añadiéndole capacidad de interacción con los mismos. Asimismo la utilización de la tecnología COM permite que se pueda acceder al simulador desde una Intranet, con lo que es posible su integración y utilización en cursos tipo e-learning. Finalmente se describe el simulador réplica desarrollado detallando los aspectos de planificación, desarrollo y pruebas. Actualmente el simulador se utiliza con éxito durante las sesiones de formación de los operadores y supervisores de la central.

2. Entorno de Simulación

El núcleo de un simulador es un programa del tipo planificador de tareas en tiempo real, que tiene la capacidad de asignar en el tiempo los distintos modelos lógicos y dinámicos de los sistemas del proceso simulado y también permite el acceso al espacio de la memoria donde la simulación se está ejecutando. La última característica se espera que se realice desde cualquier ordenador de la red de área local que soporta el simulador. El diseño del núcleo de simulación, por tanto, recoge conceptos de los sistemas operativos, comunicación entre procesos y redes de ordenadores. La plataforma de desarrollo utilizada para soportar el simulador corresponde a ordenadores personales de bajo coste bajo sistema operativo Windows, debido a su gran implantación dentro de los usuarios de ordenadores, lo que significa economía y el uso de herramientas y programas bien conocidos durante el desarrollo y mantenimiento del simulador.

Los programas que soportan el simulador se han dividido en dos grupos: el gestor del simulador y las librerías que permiten el acceso y comunicación con la simulación, y la consola del instructor. La configuración de los programas corresponden a programas cliente/ servidor, el gestor de la ejecución, los programas que atienden la entrada/salida y la consola del instructor.

La arquitectura de los programas consiste de servidores, librería cliente y las librerías de interfaz con el sistema operativo. Las aplicaciones cliente acceden a varios servicios de los servidores llamando a rutinas que usan llamadas de procedimientos remota (RPC) contenida en las librerías cliente. La librería cliente se ha diseñado para permitir que el servidor y el cliente sean procesos separados y que se ejecuten en diferentes máquinas de la red. La librería de interfaz con el sistema operativo ofrece un API a varios servicios de control de procesos usados en aplicaciones de tiempo real. Todas las librerías se han desarrollado como aplicaciones nativas de 32-bits para ser ejecutadas en el sistema operativo Windows.

La arquitectura utilizada para el entorno permite que los ordenadores conectados a una red suministren y/o utilicen servicios distribuidos dentro de un entorno cliente/servidor. Los programas a instalar en una máquina dependen de la función asignada a la misma que se pueden clasificar como: Servidor del Sistema, Consola del Instructor, Consola del Ingeniero y Ordenador del Simulador. A continuación se describen de forma resumida las principales características de cada ordenador:

- La función del Servidor del Sistema es almacenar todos los programas del simulador que incluyen las bases de datos, el código fuente de los modelos de simulación, los gráficos de la consola del instructor y los ficheros de datos. Las bases de datos se pueden gestionar y mantener usando MS SQL Server o Microsoft Access. Los demás ordenadores en la red pueden acceder al servidor de SQL mediante ODBC. Las bases de datos contienen información del servidor de directorios y de los modelos de simulación.
- El ordenador de simulación es el encargado de ejecutar el programa maestro. Los periféricos y los programas de comunicación se conectarán con este ordenador. Esto incluye los paneles reales, sistemas de apoyo al operador y otros dispositivos requeridos. Los ficheros de condiciones iniciales, vuelta atrás y los necesarios para mantener la ejecución del simulador se almacenan en este ordenador.
- El ordenador de ingeniería permite al personal de desarrollo y mantenimiento del simulador verificar

el estado del mismo. También es capaz de ejecutar la carga de simulación sin paneles. Los modelos de los sistemas se desarrollan y prueban es este ordenador utilizando normalmente como lenguajes de programación Fortran y C. Existen en el mercado herramientas de generación de modelos tales como Matlab, Modelica o EcoSim.

– La consola del instructor contiene todos los programas necesarios para que el instructor pueda visualizar y controlar el curso de la simulación. Para facilitar la gestión de la consola los programas ofrecen una interfaz Windows con elementos gráficos interactivos.

La funcionalidad de los ordenadores mencionadas anteriormente pueden agruparse en uno solo pudiendo incluso instalarse en un ordenador portátil. La configuración de los ordenadores se realiza mediante variables de entorno que pueden modificarse en tiempo real. La ejecución en tiempo real de los modelos se consigue utilizando el paradigma de programación multihilo. En todos los ordenadores se utiliza el servicio PortMapper que soporta los servicios RPC del entorno. Con esta técnica un programa puede ejecutar otro programa en una máquina remota, pasarle datos y recibir el resultado.

Adicionalmente las capacidades ofrecidas por el entorno de simulación son:

– Un sistema de ejecución que sincroniza y controla la ejecución de los modelos de simulación. A este sistema también se le llama tarea principal.
– Un sistema de base de datos para gestionar de manera centralizada los datos globales.
– Una herramienta de depuración capaz de monitorizar y modificar las variables y constantes del simulador mientras el simulador está en ejecución.
– Utilidades de precompilación que verifica la sintaxis del código fuente, valida los símbolos de la base de datos y empareja las direcciones de memoria para cada símbolo declarado en la base de datos.
– Enlazado de los módulos al sistema ejecutivo mediante el uso de librerías estáticas.

El sistema ejecutivo se encarga de asignar de forma sincronizada la ejecución de los modelos de simulación dentro de intervalos de tiempo que normalmente van de 50 a 500 mseg. Los modelos por su parte se asignan a los intervalos para mantener una carga balanceada del proceso y poder cumplir con el requerimiento de tiempo real. La información necesaria para secuenciar los modelos dentro de la carga de simulación se configura mediante un fichero de texto simple. Otra parte importante del sistema ejecutivo es el módulo correspondiente al servidor de control del simulador que se ejecuta en tiempo real y que permite el acceso a memoria de las particiones globales y usa el servicio de semáforos para controlar el sistema. Estos servicios están disponibles por cualquier cliente e incluyen funciones de detener, ejecutar, reiniciar, grabar el estado del simulador, así como atender y controlar las entradas/salidas.

Adicionalmente el sistema ejecutivo realiza las siguientes funciones:

– Configura los equipos.
– Sincroniza el entorno.
– Provee de un reloj en tiempo real para ejecutar cada módulo de control.
– Lee y escribe las condiciones iniciales y los puntos de vuelta atrás.
– Comunica los valores de los datos globales con la consola del instructor y otros dispositivos externos.

Para la programación de este sistema se ha usado lenguaje C++ y el entorno de Microsoft Visual Studio V6. La figura 1 muestra de forma esquemática los componentes del simulador.

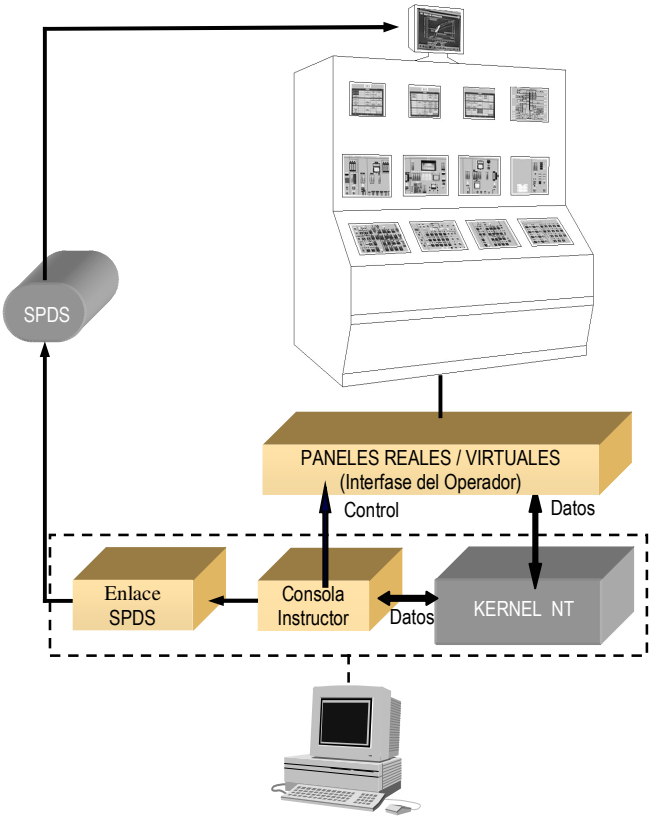


Figura 1. Esquema de los módulos del simulador.

3. Consola del Instructor

El propósito de la consola del instructor es controlar y visualizar la evolución de las variables del simulador. La consola del instructor utiliza como interfaz de usuario los diagramas de proceso e instrumentación de la simulación, menús, diálogos y ventana pop-up. A partir de los mismos el usuario es capaz de generar datos, configurar y evaluar las sesiones de entrenamiento, y controlar el simulador insertando malfunciones, funciones remotas y desalineamientos de hardware. Por tanto provee al instructor de todas las herramientas necesarias para operar y visualizar el simulador y conducir las sesiones de entrenamiento.

3.1. Operaciones de configuración del escenario

Las operaciones de configuración general más importantes disponibles desde la consola del instructor, mediante la selección de iconos de una barra de control (figura 2), son las siguientes:

Estado Detenido/Ejecución

Seleccionando el modo *Detenido* causa que la simulación deje de ejecutarse lo que produce que todos los valores se mantengan estáticos y se pare el reloj de tiempo real. Esta operación permite que la sesión de entrenamiento se interrumpa y puesta en ejecución en el mismo punto. Seleccionando el modo de *Ejecución* pone el simulador a ejecutar los modelos siguiendo el orden y asignación especificada en el fichero de módulos por el programa maestro. El simulador se pone en modo parado después de reiniciar una condición inicial o comprobación de posición de paneles.

Condiciones Iniciales

Las condiciones iniciales son los valores iniciales de las variables dinámicas de la base de datos del simulador. Estos valores se consiguen mediante cálculo o utilizando la utilidad de grabar el estado de la simulación en cualquier momento. La simulación se puede inicializar a cualquier condición inicial guardada previamente mediante este programa. Si se usan paneles reales y se establece una condición inicial es necesario comprobar que las manetas estén en la posición correcto antes de empezar la simulación. Para ello el programa avisa al instructor de los elementos desalineados para facilitar la tarea de alineamiento.

Guardar estado

Con este programa se guardan los valores en curso de las condiciones existentes en el simulador en una condición inicial para su uso posterior. Se activa mediante una pulsación del ratón cuando el simulador está en marcha.

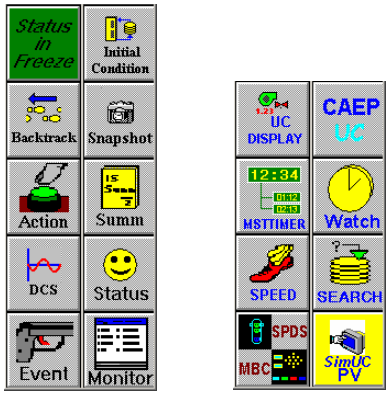


Figura 2. Barras de control del Instructor.

Reiniciar

Retorna el simulador al valor inicial determinado por una condición inicial. Este programa se activa automáticamente desde otros programas como el de condiciones iniciales, resumen, y depurador.

Vuelta atrás y repetición

Esta utilidad permite que la simulación se reinicie desde un momento previo dentro del escenario en curso. A partir de ese momento se puede reiniciar la simulación o repetir automáticamente los eventos del escenario.

3.2. Operaciones de modificación y seguimiento del escenario

Introducción de acciones del instructor: Malfunciones, funciones remotas, y alineamientos

El curso de la simulación puede ser alterado de varias maneras usando acciones del instructor. Estas acciones se pueden clasificar en tres tipos: malfunciones, funciones remotas y desalineamientos. Las malfunciones se definen como la introducción de anomalías u operaciones anormales de los componentes de la planta. Las funciones remotas, llamadas también como acciones locales del operador, son operaciones normales de planta que se realizan fuera de la sala de control. Ejemplos de funciones remotas son el cierre de una válvula manualmente, reemplazo de un fusible o arranque de un grupo diesel. Los desalineamientos simulan anomalías de los equipos de sala de control como son las manetas e indicadores. Ejemplo de este tipo de acciones son una maneta que no responde a cambios de su posición, indicador averiado, una lámpara fundida o un indicador atascado.

Todas las acciones se pueden activar de manera inmediata, retrasarlas, que varíen linealmente o asignadas a un activador. También es posible realizar

combinaciones de estos atributos. Para ello el instructor dispone de una ventana con pestañas que muestran las acciones ordenadas según un código. Dependiendo del tipo de malfunción o función remota se cambia el formato correspondiente a acciones analógicas, analógicas (normalizadas), digitales y enteras. Los alineamientos se clasifican según su tipo: salida analógica, entrada analógica, entrada digital y salida digital.

Resumen de acciones del instructor

Este programa muestra una ventana donde se resume las acciones activas y/o pendientes introducidas por el instructor. También desde esta ventana se pueden modificar las acciones.

Gráficos en tiempo real o fuera de línea

El programa de gráficos en tiempo real o fuera de línea sirve para almacenar los valores de variables seleccionadas durante la simulación o después de ella. Los gráficos se pueden configurar cambiando las escalas, etiquetas, etc. El instructor dispone de una utilidad de conversión de estos datos al formato de hojas de cálculo (csv).

Diagramas interactivos

Con este programa se obtienen los diagramas de planta o de paneles que muestran valores en tiempo real de los valores de las variables de simulación y también permite seleccionar interactivamente sobre áreas activas la introducción de acciones tales como malfunciones, acciones remotas o desalineamientos.

Activar eventos

El programa de activación de eventos sirve para iniciar acciones basadas en eventos condicionales que son expresiones definidas por el instructor. Cuando la condición se cumple, la simulación inicia la acción relacionada. Se pueden asociar varios eventos a un activador de eventos. A su vez cada activador puede tener múltiples criterios para causar que el evento se inicie.

Depuradorr o Monitor

Con este programa es posible visualizar y modificar las variables y constantes del simulador mientras está en ejecución. También se pueden introducir una cantidad limitada de comandos tales como detener/ejecutar, definir los módulos que se desean simular, introducir malfunciones, etc.

Programas dependientes de la instalación

Es posible agregar otros programas que dependen de la instalación. Por ejemplo, en el simulador desarrollado se han agregado los programas que envían las señales necesarias para el Sistema de Visualización de Parámetros de Seguridad (SPDS) y otro para el que controla el Movimiento de Barras de Control.

4. Paneles Virtuales

Antes del desarrollo del simulador réplica con paneles reales, se desarrolló un simulador de alcance total [2] cuya interacción se realizaba desde lo que denominamos paneles virtuales. Es estos paneles se reproducen mediante elementos software todos los elementos presentes en la sala de control. Para ello se desarrollaron componentes reutilizables basados en programación orientada a objetos [3] del tipo controles Active X con lo que se obtiene una gran rapidez de desarrollo de estos paneles. Los componentes se caracterizan porque se comunican directamente con el simulador y permiten la interactividad con el operador a través de zonas sensibles donde puede pulsar para producir una acción. Desde el punto de vista visual estos componentes permiten mostrar imágenes de alta calidad obtenidas con cámaras digitales, con lo que el realismo obtenido es muy alto. La figura 3 muestra ejemplos de componentes presentes en los paneles virtuales correspondientes a elementos de entrada (manetas) y de salida (indicadores).

Estos componentes se colocan de manera apropiada en una aplicación contenedora. Esta aplicación también tiene como función enviar los mensajes a los componentes cuando se realiza una nueva carga de una condición inicial o se reinicia. Tanto los componentes como los paneles se han programado con Visual C++ haciendo un uso intenso de las librerías MFC. La figura 4 muestra un ejemplo de cuatro paneles virtuales unidos para representar todo un sistema (turbina).

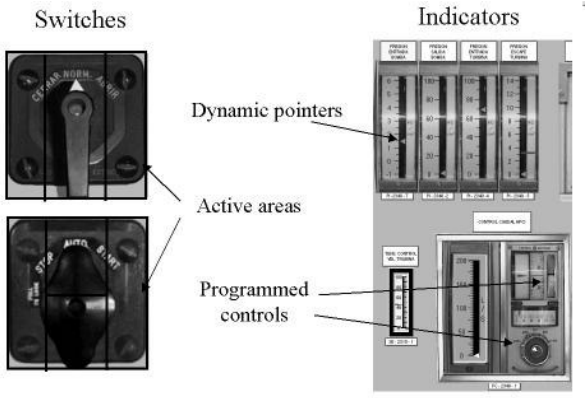


Figura 3. Tipos de componentes usados en los paneles virtuales.

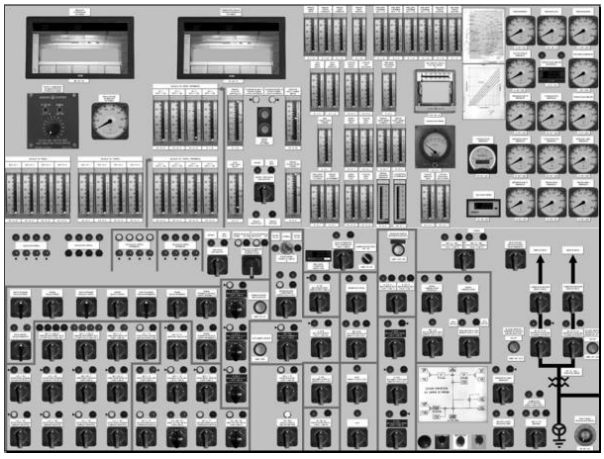


Figura 4. Composición de cuatro paneles virtuales correspondientes a los sistemas de control de turbina y generador eléctrico.

Figura 5. Un turno de operación durante una sesión de entrenamiento.

Para los paneles se pueden utilizar pantallas táctiles de alta resolución conectados a ordenadores PC de bajo rendimiento. En nuestro caso se desarrolló un soporte especial que emulaba la disposición de los paneles en la sala de control con un factor de escala del 65% y se colocaron las pantallas en él. Además se cubrieron con láminas de plástico pintadas del mismo color del soporte para incrementar la apariencia de continuidad entre los paneles y también se colocaron unas etiquetas que reproducen las existentes en la sala de control para aumentar el realismo de toda la instalación. Una vez montados todos los equipos se procedió a realizar las pruebas de validación física y funcional por parte de operadores e instructores experimentados. Después de las pruebas se entregó el simulador para el entrenamiento (2000) de los diferentes turnos de operación y para el uso del personal relacionado con la operación para la prueba de procedimientos, preparación de simulacros, etc. La figura 5 muestra una imagen durante una sesión de entrenamiento.

5. Simulación+e-Learning= SimE-Learning

Un uso novedoso de los componentes desarrollados para el simulador con paneles virtuales es que pueden ser insertados directamente en documentos preparados con MS Office, como Word, Excel o Power Point. Esta característica es posible debido a la utilización de la misma tecnología COM de programación promovida por Microsoft basada en el uso de controles ActiveX. De esta manera el instructor o persona que necesite realizar alguna operación con el simulador puede convertir un procedimiento o lección escrito en Word para el simulador, en un ejercicio dinámico de simulación añadiéndole vida al documento. La misma posibilidad de interacción se consigue con documentos preparados con Excel o PowerPoint.

A partir de esos documentos se puede generar ficheros en formato HTML para su integración en cursos de formación accesibles desde la Intranet de la empresa.

De esa forma se consigue la integración de la capacidad de simulación funcional dinámica en tiempo real con el mundo e-Learning. Además de los conceptos teóricos aportados por los temas descritos en un curso de formación, se incluye un apartado especial de "Prácticas en planta" que comprende una serie de escenarios pre-programados que se pueden activar desde una página principal. Cada escenario viene precedido por una introducción en el que se fijan los objetivos más importantes del escenario y la documentación que puede ser accedida (POEs, CWDs, TDGs, IOPs, etc.). La interacción con el simulador se establece mediante varias posibilidades de navegación: paneles virtuales o diagramas P&ID sensibles. Otros usuarios pueden acceder concurrentemente al mismo escenario con lo que sólo se puede ejecutar un escenario a la vez. Uno de ellos puede actuar de instructor para lo cual hay páginas seguras dedicadas y diseñadas para esta función. En cada sesión el simulador almacena todas las acciones llevadas a cabo durante un escenario con lo que el alumno o instructor puede comprobar si se han seguido correctamente los procedimientos.

Con estas posibilidades se incrementa la experiencia adquirida por el personal involucrado en la operación de la planta en menor tiempo, mejorando la calidad de su formación. El alumno dispone de gran flexibilidad al ser capaz de utilizar el material en cualquier

momento y en cualquier lugar. El instructor puede desarrollar módulos de entrenamiento personalizados.

6. Simulador réplica

Con la experiencia obtenida en el desarrollo del simulador con paneles virtuales se desarrolló el simulador réplica de alcance total. Para ello se ampliaron y mejoraron una serie de modelos de sistemas que abarcan la totalidad de los existentes en la planta real. El entorno de simulación se amplió para incluir la comunicación con los paneles reales, basado principalmente en el protocolo TCP/IP. Además de los paneles frontales (figura 5) se han simulado los paneles traseros y el panel de parada remota. En los paneles traseros se han simulado una serie de controladores, indicadores y registradores haciendo uso de paneles virtuales. El desarrollo se realizó en 18 meses por un equipo de cinco analistas. Durante el desarrollo se siguieron las estrictas normas de calidad aplicables al software en la industria nuclear, utilizando como herramienta de control de configuración de los códigos y documentación MS Source Safe, que viene dentro de los programas de Visual Studio. Después de superar las pruebas de aceptación y licenciamiento, el simulador se encuentra en operación desde finales de 2003.

Figura 5. Disposición de los paneles frontales del simulador réplica.

6. Modelo de desarrollo

Para el desarrollo de un simulador réplica es habitual adoptar un modelo de desarrollo tradicional de tipo cascada. Los requerimientos básicos que debe cumplir todo simulador se encuentran fijados en la norma ANSI [4] a la que se agregan los requerimientos específicos de la instalación que se va a simular. Para el simulador réplica desarrollado se consideró que debían representarse todos los paneles frontales, traseros y del panel de parada remota, es decir, un alcance superior a otros simuladores. En los paneles traseros y el panel de parada remoto se simularon algunos registradores y controles mediante paneles virtuales, debido a una menor exigencia de fidelidad física. Los requerimientos mínimos en un simulador réplica son de dos tipos: fidelidad física y funcional. El primero de ellos (fidelidad física) es de naturaleza hardware y consiste en que la sala de control del simulador debe ser idéntica a la real. La fidelidad funcional es de naturaleza software y consiste en que el comportamiento de los modelos de los sistemas, instrumentación, controles, comunicaciones, etc. debe ser lo más parecido posible al real dentro de una tolerancia muy baja, de manera que el alumno perciba un comportamiento muy similar al real.

Para el desarrollo del entorno de simulación se utilizó un modelo de desarrollo evolutivo basado en prototipos y para los paneles virtuales se utilizó un modelo de desarrollo basado en la reusabilidad de componentes. En el primer caso se desarrollaron un conjunto de librerías que permitieron el refinamiento y mejora del entorno de simulación, siendo los requerimientos de tiempo real, acceso distribuido y gestión de la base de datos de señales las más importantes. En el segundo caso se desarrollaron componentes para los indicadores, registradores y controles a partir de los cuales se integraron según el panel.

7. Conclusiones

La seguridad de los procesos industriales pasa por un buena formación del personal involucrado en la operación. En la industria nuclear es primordial la formación y el entrenamiento de los operadores en todo tipo de situaciones. Para conseguir ese objetivo la formación de los operadores se realiza en simuladores réplica de alcance total, es decir simuladores que reproducen con un alto grado de fidelidad física y funcional todos los elementos presentes en la sala de control. El proyecto de desarrollo de ese tipo de instalaciones es complejo e involucra personal con experiencia en diversas áreas como modelado, programación en Windows y comunicaciones. La tecnología desarrollada para realizar un simulador con paneles virtuales se ha aprovechado para aplicarla al desarrollo de un simulador réplica de alcance total. Un buen diseño del entorno de simulación juega un papel muy importante en la eficiencia del simulador. Al utilizar la plataforma PC con sistema operativo Windows como soporte del simulador se obtiene un producto muy económico. El uso de componentes ActiveX que pueden integrarse en documentos MSOffice, permite que se puedan acceder desde

IExplorer consiguiendo su uso en cursos de formación del tipo e-Learning.

8. Referencias

[1] P. Corcuera, F. Bustío, E. Mora, "Training Simulator for Garoña Nuclear Power Plant", *Lecture Notes in Computer Science*, Springer Verlag, Vol. 1030, 1996, pp. 523-529.

[2] P. Corcuera, "A Full Scope Nuclear Power Plant Training Simulator: Design and Implementation Experiences", *Journal of Systemics, Cybernetics and Informatics*,. Vol. 1, Number 3, 2003, pp. 12-17.

[3] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*, Addison-Wesley, 1998..

[4] ANSI/ANS-3.5-1998, *Nuclear Power Plant Simulators for Use in Operator Training and Examination.*

Avaliando a Relação entre Tamanho-Complexidade e Número de Defeitos de Software em Nível de Módulo

Waldo Luis de Lucca
UNIMEP – Universidade Metodista de Piracicaba
wllucca@unimep.br

Plínio R. S. Vilela
UNIMEP – Universidade Metodista de Piracicaba
prvilela@unimep.br

Mario Jino
UNICAMP – Universidade Estadual de Campinas
jino@dca.fee.unicamp.br

Abstract

A challenge in software testing research area is presented by the question: how do we reduce time and effort to test software without reducing the effectiveness? In this paper, we present the results of an experiment with a program that contains real defects. The data collection have been performed in module-level, rather than statement-level, such as what was done in a previous work. Our primary hypothesis is that the fault-revealing rate is greater when larger or more complex modules are tested first. We demonstrate that testing modules ordered by size or complexity metrics reduce the number of modules to be tested in 6,2% to 17,7%. A secondary hypothesis was formulated: a larger or more complex module has higher probability of containing a defect than smaller or less complex module. We observed that the probability of a module containing a defect is 43,8% to 78,7% greater in modules with size or complexity above the average. Our empirical results support both hypothesis and are an important reference to new experiments and studies that intend to develop new techniques and strategies to reduce the cost of software testing.

Resumo

Um desafio de pesquisa na área de teste de software é apresentado na questão: como podemos reduzir o tempo e esforço para testar um software sem reduzir sua eficácia? Neste trabalho, nós apresentamos os resultados de um experimento com um programa que contém defeitos reais documentados. A coleta de dados foi feita em nível de módulo. Em trabalho anterior, já havíamos feito em nível de segmentos de código. Nossa hipótese principal é que a taxa de detecção de defeitos é maior se os módulos maiores ou mais complexos forem testados primeiro. Nós demonstramos que testando os módulos ordenados por métricas de tamanho ou complexidade pode-se reduzir em 6,2% até 17,7% o número de módulos a serem testados. Uma hipótese secundária foi formulada: um módulo maior ou mais complexo tem maior probabilidade de conter um defeito do que módulos menores ou menos complexos. Nós observamos que a probabilidade de um módulo conter defeito é de 43,8% a 78,7% maior em módulos com tamanho ou complexidade acima da média. Os resultados de nossos experimentos suportam ambas as hipóteses e são referências importantes para novos experimentos e estudos que buscam o desenvolvimento de novas técnicas e estratégias para reduzir os custos do teste de software.

1. Introdução

Um dos desafios do teste de software é identificar o maior número de defeitos no menor tempo e com o menor custo possível. A relação entre o número de defeitos identificados pelo teste de software e o tempo e custo relacionados é um tema que precisa ser mais investigado para que seja possível a proposta de técnicas que aumentem a eficácia da atividade de teste.

Várias técnicas de teste foram propostas ao longo do tempo e pode-se afirmar que elas surgem como uma alternativa de busca da melhor relação possível entre seu custo e seus benefícios para a qualidade de software. A atividade de teste não pode garantir a ausência de defeitos no software [6] e seu objetivo principal é revelar a presença de defeitos [13]. Somente seria possível garantir a total ausência de defeitos em um software se todos os seus caminhos pudessem ter sido exercitados durante o teste, com todas as combinações possíveis de dados de entrada, dentro de todas as possibilidades de ocorrências no ambiente de software e hardware em que o programa está inserido.

Essa situação de teste exaustivo é impossível para a quase totalidade de software que se tem desenvolvido ao longo da história.

Como o teste exaustivo não pode ser considerado na maioria dos softwares produzidos, as técnicas de teste disponíveis atualmente podem contribuir para a definição de um conjunto mínimo de requisitos que devem ser testados para que se possa aferir alguma garantia de qualidade. Mas nem sempre estas técnicas são utilizadas em função de que ainda são caras e tomam muito tempo do processo de desenvolvimento do software. A ausência da aplicação de técnicas de teste tem feito com que muitos softwares sejam produzidos com baixa qualidade, não funcionando da forma como deveriam funcionar.

Uma questão importante para a redução do custo da atividade de teste é pensar quais são os aspectos que realmente influenciam a existência dos defeitos no software. Este trabalho apresenta um experimento e busca analisar dados referentes à localização dos defeitos em módulos de software.

O objetivo deste trabalho é contribuir para o conhecimento acerca da relação entre defeitos e esforço de teste, em busca de desenvolvimento de técnicas e estratégias de teste que permitam a redução de seu custo, sem perder em eficácia na detecção de defeitos.

A seção seguinte apresenta alguns conceitos relacionados à atividade de teste e sua relação com métricas de tamanho, complexidade e número de defeitos. A Seção 3 descreve o experimento realizado e o programa que foi utilizado no experimento. A Seção 4 discute alguns resultados percebidos durante a realização do experimento. E, finalmente, a Seção 5 apresenta as conclusões tiradas a partir do estudo realizado e aponta trabalhos futuros.

2. Métricas de tamanho e complexidade e número de defeitos

O esforço para a realização da atividade de teste deve levar em consideração vários fatores do processo e características do programa a ser testado. As atividades que compõem o processo de teste, o nível em que o teste é realizado e os critérios aplicados, dentre outros aspectos, influenciam o esforço e o custo da atividade e seu impacto no custo total do processo de desenvolvimento.

Dentre os aspectos que influenciam o esforço e custo do teste de software está o tempo gasto na atividade de teste, incluindo o planejamento dos testes, a geração de casos de teste, a execução do programa com o conjunto de casos de teste gerado e análise dos resultados. Vários experimentos foram realizados buscando-se identificar se há alguma relação entre o tamanho do software, medido em linhas de código (LOC – *Lines Of Code*) e o tempo gasto para testá-lo. Ellims et al. [8] chegaram a uma variação entre 2,5 a 15 LOC por hora de teste. Essa variação está relacionada, basicamente, ao tipo de projeto e tipo de processo.

O tempo gasto para executar o teste está também relacionado ao número de requisitos a serem testados em um programa. Se considerarmos apenas esse aspecto, quanto mais requisitos forem testados maior tempo será necessário para a conclusão dessa atividade. Assim, uma forma de reduzir o tempo de teste é reduzir o número de requisitos a serem testados. Porém, essa redução pode significar uma perda de eficácia da atividade de teste, uma vez que é possível, ao reduzir o número de requisitos de teste, reduzir também o número de defeitos revelados.

Um dos desafios da área de teste de software é reduzir o tempo de execução dessa atividade sem reduzir a eficácia na detecção de defeitos. Nesse sentido, uma questão central é identificar em que partes do software os defeitos geralmente se encontram. Não há uma resposta definitiva sobre essa questão. Ao mesmo tempo em que se pode supor que as partes maiores ou mais complexas do software têm mais defeitos que as partes menores ou mais simples, é também possível entender que as partes maiores ou mais complexas, pelas suas peculiaridades, são programadas com maior cuidado e, conseqüentemente, têm menor número de defeitos.

Vários modelos de predição de defeitos têm sido propostos baseados em tamanho e complexidade [1, 7, 9, 10, 15, 16]. Compton e Withrow [3] e Moeller e Paulish [12] analisam a influência do tamanho do programa em LOC na densidade de defeitos, em vez do número de defeitos, concluindo que a densidade de defeitos é menor em módulos maiores.

Métricas de tamanho e complexidade estão sendo consideradas neste trabalho, pois há evidências, pelos trabalhos citados anteriormente, que essas métricas são boas para predizer o número de defeitos do software. Em experimento anterior, buscamos identificar a adequação de um conjunto de métricas de tamanho e complexidade para predizer o número de defeitos de um software, em um nível de granularidade menor do que módulos, ou seja, em segmentos de código que implementam requisitos de teste baseado em fluxo de dados (associações entre definição e uso de variáveis do programa) [14]. Os resultados contribuem para essa constatação.

Dentre as métricas citadas na literatura, LOC é a mais utilizada nos modelos de predição de defeitos. Neste trabalho, LOC é usada como uma métrica de tamanho. Embora muitas vezes usados para a mesma finalidade, fazemos uma distinção entre tamanho e complexidade, por entender que um módulo com maior número de linhas de código do que outro não significa que necessariamente é mais complexo do que outro. Há certas características de programação que requerem muitas linhas de código em determinadas linguagens, mas com a utilização de poucas estruturas de controle e lógica simples. Há também módulos pequenos, mas com complexidade alta. Uma linha de código pode ser mais ou menos complexa que outra linha de código.

Dentre as métricas de complexidade, uma que é adequada para o nível de módulo é a complexidade ciclomática de McCabe [11]. Essa métrica baseia-se no grafo de fluxo de controle correspondente ao programa e indica o grau de sua complexidade. Valores maiores para a complexidade ciclomática indicam maior complexidade. A complexidade ciclomática pode ser definida como o número de nós predicativos (ou número de comandos de decisão) do programa, acrescido de um.

O número de mutantes de um programa também pode ser considerado uma medida de complexidade. Pela técnica de análise de mutação [5], mutantes são variantes do programa principal, contendo uma única alteração sintática em relação ao programa original. As alterações contidas nos programas mutantes são geradas a partir de operadores de mutação, e correspondem a pequenos defeitos comuns de programação. Consideramos que o número de mutantes pode ser útil para medir a complexidade de um programa, uma vez que os operadores de mutação estão associados com estruturas de programação. Quanto mais complexo um programa, maior será o número de mutantes gerados a partir de sua estrutura.

Usando as métricas de tamanho e complexidade e a afirmação de que um programa maior ou mais complexo pode conter mais defeitos, poderia-se considerar que quanto maior o número de linhas de código, ou a complexidade ciclomática ou o número de mutantes de um programa ou módulo, maior o número de defeitos esperado. Porém, como visto anteriormente, é possível que não exista uma relação linear entre tais métricas e o número de defeitos. Dessa forma, há lugar para investigar se as métricas de tamanho e complexidade podem ser usadas para, de alguma maneira, indicar o que poderia ser testado primeiro, de forma a revelar os defeitos em menor tempo e, conseqüentemente, reduzir o custo de sua execução.

O experimento que será relatado a seguir buscou avaliar se o uso de métricas de tamanho e complexidade servem para indicar a ordem pela qual os módulos devem ser testados, de forma que se teste todos os módulos com defeitos mais cedo do que uma ordem aleatória. Os resultados apresentam algumas indicações de que esse pode ser um caminho a ser utilizado em técnicas e estratégias para a redução do custo da atividade de teste, sem perder sua eficácia.

3. Descrição do Experimento

O presente trabalho é baseado em um experimento que utilizou a seguinte metodologia para sua realização:

1. Definição da hipótese
2. Escolha do programa
3. Escolha das métricas
4. Coleta de dados
5. Teste da hipótese

A discussão sobre os modelos de predição de defeitos, sinteticamente apresentada na seção anterior, e a observação de alguns aspectos da atividade de programação, nos faz pensar que um módulo maior ou mais complexo teria mais chance de conter defeitos do que um módulo menor e menos complexo. Embora não se imagine que há uma relação linear entre o tamanho ou complexidade e o número de defeitos, é possível pensar que um teste realizado inicialmente sobre os módulos maiores ou mais complexos pode levar à revelação de todos os defeitos conhecidos de um software mais cedo do que um teste sobre todos os módulos em ordem aleatória.

A partir dessa observação, formulamos a hipótese principal deste trabalho, que é a de que o tempo necessário para a revelação dos defeitos de um software é menor quando os módulos maiores ou mais complexos são testados primeiro. Por tratar-se de um experimento controlado, é importante salientar que os defeitos do software e sua localização são conhecidos, pelo fato do programa já ter sido testado anteriormente. Embora possa ser estimado, o número de defeitos geralmente não é conhecido quando se inicia a atividade de teste de um software.

A hipótese principal deste trabalho implica em que é possível haver uma redução no tempo de execução do teste, e conseqüentemente do seu custo, se métricas de tamanho e complexidade forem utilizadas para definir o que testar primeiro em um software.

A hipótese formulada também permite, se confirmada, auxiliar na previsão de defeitos do software e do tempo e esforço necessário para testá-lo até que os defeitos sejam revelados.

Uma hipótese secundária é de que há maior probabilidade de conter defeitos nos módulos maiores e mais complexos do que nos menores e menos complexos.

O experimento realizado utilizou o software Space, um programa desenvolvido pela Agência Espacial Européia (European Space Agency), na linguagem C. Esse programa provê uma interface que permite o usuário descrever a configuração de um conjunto de antenas utilizando uma linguagem de alto nível [2]. A escolha desse programa se deu pelo fato de ter defeitos reais documentados, além de já ter sido usado em outros experimentos [17].

A versão utilizada do programa inclui a documentação de 34 defeitos reais que foram detectados durante os testes. Foram utilizados, no experimento, 134 módulos do software, que totalizam 4.367 linhas de código. Os 34 defeitos reais estão localizados em 25 módulos do programa.

O escopo do experimento incluiu apenas algumas variáveis, como o tamanho e a complexidade dos módulos e o número de defeitos. Para esse objetivo, intencionalmente não foram consideradas as diferenças de esforço no teste de cada módulo, nem o fato de que o teste de um módulo com defeitos não garante que esses defeitos sejam revelados.

As métricas selecionadas para o experimento, além do número de defeitos, foram: linhas de código (LOC), complexidade ciclomática de McCabe [11] e número de mutantes [5]. LOC foi utilizada como métrica de tamanho, enquanto que complexidade ciclomática e número de mutantes foram usadas como métricas de complexidade.

A contagem do número de linhas de código foi feita por módulo, desconsiderando as linhas em branco e linhas exclusivamente de comentários. A complexidade ciclomática (V(G)) foi calculada a partir do número de comandos de decisão do programa, mais um. O número de mutantes de cada módulo foi calculado utilizando-se uma ferramenta chamada Proteum [4]. Os dados coletados foram transferidos para uma planilha de cálculo, onde o teste da hipótese foi realizado.

Para se realizar um experimento dessa natureza precisamos utilizar um software com defeitos conhecidos, uma vez tendo-se a hipótese confirmada podemos aplicar os resultados na definição de estratégias e técnicas de teste para programas dos quais não conhecemos o número total nem a localização dos defeitos. Para testar a hipótese definiu-se como critério de avaliação o número de módulos a serem testados até que todos os defeitos conhecidos do software fossem revelados. Se o teste dos módulos do programa numa certa ordem de tamanho ou complexidade permitisse revelar 100% dos defeitos conhecidos antes do que o teste dos mesmos módulos em ordem aleatória, então a hipótese seria considerada aceita.

O teste da hipótese utilizou fundamentos estatísticos de probabilidades e análise de correlação. Foi calculada, inicialmente, a probabilidade de um módulo do programa conter defeitos. Em seguida foram calculadas, para cada métrica, seu valor médio, que serviu de referência para o cálculo de novas probabilidades de ocorrência de módulo com defeitos, distinguindo-se entre os módulos com valor da métrica acima da média e os módulos com valor da métrica abaixo da média. Essas probabilidades foram usadas para avaliação da hipótese secundária de que há maior probabilidade de conter defeitos nos módulos maiores e mais complexos do que nos menores e menos complexos.

Para o teste da hipótese principal, foi calculado o número de módulos que provavelmente teriam que ser testados até que o último defeito conhecido fosse revelado numa ordem aleatória, e esse número foi comparado com o número de módulos que seriam necessários testar até encontrar o último defeito conhecido na ordem de cada uma das métricas de tamanho e complexidade utilizadas. Também foi utilizada análise de correlação, para avaliar a correspondência das métricas utilizadas e o número de defeitos.

Para se chegar ao número de módulos que provavelmente teriam que ser testados até que o último defeito conhecido fosse revelado numa ordem aleatória, foi utilizado um programa para gerar 40.000 listas em ordem aleatória de módulos, identificando-se, em cada lista, a posição correspondente do último módulo com defeito. O número de listas geradas aleatoriamente poderia ter sido menor, mas como o processamento do programa que as gerou foi eficiente, optamos por gerar um número maior de listas para obter uma maior precisão no número de módulos necessários até chegar ao último módulo com defeito.

O número de módulos que teriam que ser testados até que o último defeito conhecido fosse revelado em ordem de tamanho e complexidade foi definido a partir do ordenamento das listas de módulos por LOC, V(G) e número de mutantes, do maior para o menor, considerando o índice de cada lista em que o último módulo com defeitos seria testado.

4. Discussão dos Resultados

Para o programa analisado, com seus 34 defeitos e 134 módulos, a probabilidade de um módulo conter defeito é de 0,187. Essa probabilidade aumenta, em

relação a todas as métricas utilizadas no experimento, se forem considerados somente os módulos cujos valores das métricas estiverem acima de suas médias.

A média do tamanho dos módulos considerados é 32,6 LOC. Somente 48 dos 134 módulos têm tamanho acima da média, correspondente a 35,8% do total de módulos. Se considerado apenas esse subconjunto, a probabilidade de um módulo conter defeito aumenta para 0,313.

Em termos de complexidade, o programa usado no experimento apresentou V(G) médio de seus módulos igual a 5,3. Do total, apenas 41 módulos têm complexidade ciclomática acima da média, ou seja, 30,6%. Nesse caso, a probabilidade de um módulo com defeito no grupo de complexidade acima da média é de 0,268. Em relação ao número de mutantes, a média dos módulos é de 784,3 mutantes. Dos 134 módulos, 33, ou 24,6%, possuem número de mutantes maior que a média. A probabilidade de um módulo com defeito nesses 33 módulos é de 0,333.

Os dados apresentados acima comprovam, para o programa em questão, a hipótese secundária, de que há maior probabilidade de conter defeitos nos módulos maiores e mais complexos do que nos menores e menos complexos. Em todos os casos, houve aumento da probabilidade de um módulo conter defeito a medida em que foram considerados subconjuntos formados apenas pelos módulos cujos valores da métrica estavam acima de suas médias. Esse aumento variou de 43,8% a 78,7% em relação à probabilidade de um módulo conter defeito sem distinção de tamanho ou complexidade. A Tabela 1 apresenta as probabilidades em cada caso.

Tabela 1. Probabilidades de módulos com defeitos.

Módulos considerados	Probabilidade	%
Módulos com defeito (total)	0,187	100,0%
Módulos com LOC acima da média	0,313	167,5%
Módulos com V(G) acima da média	0,268	143,8%
Módulos com número de mutantes acima da média	0,333	178,7%

Para se confirmar a hipótese principal deste estudo, de que o tempo necessário para a revelação dos defeitos conhecidos de um software é menor quando os módulos maiores ou mais complexos são testados primeiro, os módulos do programa foram colocados em ordem decrescente de tamanho ou complexidade e foram calculados alguns valores de referência para a comparação, desprezando-se as diferenças de esforço para se testar cada um dos módulos. Esses valores de referência são o número de módulos que deveriam ser testados em três situações: (1) se todos os módulos fossem testados; (2) se apenas os módulos com defeito fossem testados; (3) se fossem testados, em ordem aleatória, a quantidade de módulos até aquele que contém o último defeito conhecido.

Se todos os módulos fossem testados, seria necessário testar os 134 módulos. Esse seria o teste mais demorado e custoso. A solução ótima seria testar apenas os módulos com defeito. Nesse caso, bastaria testar 25 módulos, com uma economia de esforço de 81,3%. Caso nenhum critério de ordenamento dos módulos no teste fosse usado, o número médio de módulos que deveriam ser testados até que o último módulo com defeito fosse testado é 130 módulos, representando uma economia de 3% em relação ao teste de todos os módulos. Esse número de módulos foi definido pela média da localização do último módulo com defeito dentre um conjunto de 40.000 listas geradas aleatoriamente.

Os módulos foram ordenados segundo as métricas utilizadas e, aplicando-se cada uma dessas ordens, foram obtidos o número de módulos que seriam necessários testar para que, segundo cada ordem, atingir o módulo contendo o último defeito conhecido. A Tabela 2 apresenta esses valores, em comparação com os valores de referência.

Tabela 2. Número de módulos necessários de serem testados até que o último módulo com defeito seja atingido.

Ordem	Número de módulos a serem testados	% redução em relação a todos os módulos	% redução em relação à ordem aleatória
Todos os módulos	134	-	-
Somente módulos com defeitos	25	81,3%	80,8%
Ordem aleatória	130	3,0%	-
Ordem por LOC	107	20,1%	17,7%
Ordem por V(G)	122	9,0%	6,2%
Ordem por número de mutantes	111	17,2%	14,6%

Os dados apresentados na Tabela 2 mostram que a ordem que mais se aproxima da solução ótima é a

classificação dos módulos por LOC, representando, em média, 17,7% menos módulos a serem testados do que a ordem aleatória. A classificação dos módulos por número de mutantes também representa uma redução (14,6%), embora um pouco menor do que a ordem de LOC. A ordem de V(G) foi a que obteve menor redução em relação à ordem aleatória, representando 6,2% de redução.

Nas 40.000 listas geradas aleatoriamente neste experimento, em apenas 0,135% a ordem aleatória foi melhor (apresentou todos os módulos com defeito antes) que a ordenada por LOC, em 0,405% foi melhor que a ordem por número de mutantes e em 5,993% foi melhor que a lista ordenada por V(G).

Uma análise complementar foi feita na busca de confirmação da correlação existente entre as listas na ordem dos valores de cada métrica e o número de módulos com defeitos. Para isso, em cada lista ordenada calculou-se o valor acumulado da métrica respectiva e, na mesma ordem, o número de módulos com defeito acumulado. Nos conjuntos de dados resultantes, foram calculados os coeficientes de correlação, apresentados na Tabela 3.

Tabela 3. Coeficientes de correlação entre os valores acumulados das métricas e o número acumulado de módulos com defeito, considerando-se as listas ordenadas por LOC, V(G) e número de mutantes.

Métrica	coeficiente de correlação
LOC	0,970201
V(G)	0,970180
Número de mutantes	0,952058

Com base nos dados acima, pode-se considerar que há uma forte associação entre os valores das métricas de tamanho e complexidade usadas neste experimento e o número de módulos com defeitos.

Considerando que as classificações dos módulos pelas métricas LOC e número de mutantes apresentam a probabilidade de redução, de 12,4% e 9,2% respectivamente, no número de módulos necessários para serem testados até que o último módulo contendo defeito seja testado, e considerando que essas métricas possuem uma correlação alta com o número de módulos com defeitos, é possível afirmar que a hipótese primária deste trabalho, para o programa em questão, é verdadeira, ou seja, que o tempo necessário para a revelação dos defeitos conhecidos de um software é menor quando os módulos maiores ou mais complexos são testados primeiro, pois há maior probabilidade de revelar 100% dos defeitos conhecidos mais cedo se o teste for feito nos módulos em ordem de tamanho ou complexidade.

5. Conclusões

Este trabalhou buscou contribuir para o conhecimento acerca da relação entre tamanho, complexidade e número de defeitos, por meio de um experimento, em nível de módulo, com um programa que contém defeitos reais documentados.

Vários modelos de predição de defeitos baseiam-se em métricas de tamanho e complexidade. O experimento realizado confirmou, para o programa em questão, que os defeitos localizam-se, em maior quantidade, nos módulos maiores ou mais complexos.

Os resultados apontam que os módulos que contém defeitos podem ser testados antes se esses módulos forem ordenados por métricas de tamanho (LOC) ou complexidade (complexidade ciclomática ou número de mutantes). No caso em questão, isso pode significar uma redução de até 17,7% no número de módulos necessários para que se revelem todos os defeitos.

Os dados apresentados mostram, também, que há uma maior probabilidade de ocorrência de defeitos em módulos maiores ou mais complexos, A probabilidade de existência de defeito em um módulo acima do tamanho médio de todos os módulos do programa chega a ser 67,5% maior do que a probabilidade média do conjunto dos módulos. Em relação à complexidade, esse aumento de probabilidade é maior ainda, chegando até uma probabilidade de 78,7% maior que a probabilidade média.

Novos estudos devem ser feitos, para confirmar os resultados obtidos com o programa considerado no experimento realizado. Nesses novos estudos, outros componentes de custo devem ser considerados, como esforço de teste por módulo, requisitos de teste, aspectos do programa que podem aumentar ou reduzir a testabilidade, dentre outros, visando ao estabelecimento de técnicas e estratégias de redução de custo que levem em consideração os atributos do próprio software. Para isso, novos experimentos estão sendo planejados, incluindo a replicação do estudo apresentado neste trabalho sobre outros programas.

Referências Bibliográficas

[1] F. Akiyama, "An Example of Software System Debugging", *Proceedings of the 5th International Federation of Information Processing Congress*, agosto de 1971, pp. 353-358.

[2] A. Cancellieri, e A. Giorgi, "Array PreProcessor User Manual", *Relatório Técnico IDS-RT94/052*, 1994.

[3] T. Compton, e C. Withrow, "Prediction and Control of Ada Software Defects", *Jornal of Systems and Software*, v. 12, n. 3, julho de 1990, pp. 199-207.

[4] M. Delamaro, J. C. Maldonado, M. Jino, e M. Chaim, "Proteum: A Testing Tool Based on Mutation Analysis", *7th Brazilian Symposium on Software Engineering*, outubro 1993.

[5] R. A. DeMillo, R. J. Lipton, e F. G. Sayward, "Hints on Test Data Selection: Help for the Practicing Programmer", *IEEE Computer*, vol. 11, n. 4, abril de 1978, pp. 34-41.

[6] E. W. Dijkstra, "Notes on Structured Programming", In: O. J. Dahl, E. W. Dijkstra, e C. A. R. Hoare, *Structured Programming*, Academic Press, 1972.

[7] J. R. Gaffney, "Estimating the Number of Faults in Code", *IEEE Transactions on Software Engineering*, v. 25, n. 5, setembro/outubro de 1999, pp. 675-689.

[8] M. Ellims, J. Bridges, e D. C. Ince, "The Economics of Unit Testing", *Empirical Software Engineering*, vol. 11, n. 1, março de 2006, pp. 5-31.

[9] M. H. Halstead, *Elements of Software Science*, Operating and Programming Systems Series, Elsevier Science, 1977.

[10] M. Lipow, "Number of Faults per Line of Code", *IEEE Transactions on Software Engineering*, v. 10, n. 4, julho/agosto 1982, pp. 437-439.

[11] T. J. McCabe, "A Complexity Measure", *IEEE Transactions on Sfotware Engineering*, v. 2, n. 4, dezembro 1976, pp. 308-320.

[12] K. H. Moeller, e D. Paulish, "An Empirical Investigation of Software Fault Distribution", *Proceedings of 1st International Software Metrics Symposium*, maio de 1993, pp. 82-90.

[13] G. Myers, *The Art of Software Testing*, Willey, 1979.

[14] P. R. S. Vilela, W. L. Lucca, A. Corso, e M. Jino, "Comparison of Size and Complexity Metrics as Predictors of the Number of Faults", *Anais da IV Jornadas Iberoamericanas en Ingeniería del Software e Ingeniería del Conocimento*, v. I, Madrid - Espanha, 2004, pp. 633-644.

[15] W. E. Wong, J. R. Horgan, S. London, e A. Mathur, "Effect of Test Set Minimization on the Fault Detection Effectiveness of teh All-Uses Criterion". *Relatório Técnico*, Purdue University, 1994.

[16] W. E. Wong, J. R. Horgan, S. London, e A. Mathur, "Effect of Test Set Minimization on Fault Detection Effectiveness", *Software – Practice and Experience*, v. 28, n. 4, abril de 1998, pp. 347-369.

[17] W. E. Wong, J. R. Horgan, A. P. Mathur, and A. Pasquini, "Test Set Size Minimization and Fault Detection Effectiveness: A Case Study in a Space Application", *The Journal of Systems and Software* , n. 48, 1999, pp. 79-89.

Empirically Evaluating the Usefulness of Software Visualization Techniques in Program Comprehension Activities

Glauco de F. Carneiro, Angelo C. Araujo Orrico and Manoel G. de Mendonça Neto
Software Engineering and Applications Group (GESA/NUPERC)
Salvador University – UNIFACS
glauco.carneiro@unifacs.br, angelo.orrico@si.unifacs.br, mgmn@unifacs.br

Abstract

Program comprehension is the basis for software maintenance and reengineering. It is possible and viable the use of information visualization techniques to improve the level of program comprehension through software visualization. This paper presents an empirical evaluation on the usefulness of software visualization techniques to program comprehension activities from the viewpoint of software maintainers. In the experiment, ten students execute a set of program comprehension activities with the assistance of a software visualization tool, called SourceMiner, over a set of two open source software systems. The results show that when using SourceMiner coupled with an IDE, as opposed to the IDE alone, the students found 45% more restructuring and bad smells opportunities in the analyzed code.

Key words: Program Comprehension, Software Visualization, Source code metrics, Treemaps.

1 Introduction

Software comprehension is the basis for software maintenance activities. Maintenance is associated with a representative percentage of the software life cycle costs [11]. In order to accomplish software comprehension goals, one needs to deal with, understand and manage several kinds of artifacts, notably source code. Extracting information from industry strength software systems is difficult due to their size and complexity. The success of program comprehension depends on the experience and background knowledge of the group that performs this activity as well as the techniques and approaches used by them.

The cognitive process of human beings is more successful when supported by visual resources such as images, drawings and signs. Textual and tabular representations usually show drawbacks. As a consequence, costs associated with activities based on software comprehension can be decreased when supported by visual resources [1].

This paper reports and analyzes the results of an experiment to evaluate the use of a software visualization technique called *treemaps* in program comprehension activities.

Treemaps were proposed by Ben Shneiderman during the 1990s with the idea of producing a compact visualization of file directories [13]. From then on, it has been used to present structural hierarchy of information in domain areas such as Bioinformatics, News, Elections, Finance, Marketing, Management and Reporting [13]. They are used for space-constrained visualization of hierarchical structures, and are very effective in showing large hierarchical structures in the limited bi-dimensional space of a computer screen. For that, it uses recursively nested rectangles to represent the hierarchy. Attributes of the hierarchy leaf nodes can be nicely mapped to the size and color of the rectangles. Treemaps enable users to compare nodes and sub-trees even at varying depths of the hierarchy, and it is very effective in highlighting patterns and exceptions [12].

Our work uses treemaps to represent and explore large volumes of Java source code. In this study, systems of up to 75 thousands of lines of code have been mapped to interactive treemaps in a tool called SourceMiner. The tool shows the code structure as a hierarchy of packages, classes and methods. Source code metrics such as cyclomatic complexity and size can be visually shown as the rectangles color and size.

This paper is organized as follows. Section 2 explains traditional approaches applied to program comprehension. Section 3 presents the advantages of adopting visual metaphors in program comprehension activities and the *SourceMiner* tool. Section 4 discusses an experiment that evaluates the usefulness of visualization in program comprehension activities. Section 5 presents and discusses the experimental

results. Section 6 provides conclusions and discusses future work.

2 Software Comprehension

Based upon software psychology studies, a variety of models have been proposed for the process of program comprehension. Program comprehension is a process that uses existing knowledge to acquire new knowledge that ultimately meets the goals of a code cognition task [6]; or a deductive process of acquiring knowledge about a software artifact through analysis, abstraction, and generalization [7]. The knowledge acquired during the program comprehension process guides the programmer to recognition of abstract items, like the control structure of the program, its global architecture and functionalities. Program comprehension relies on the source code as the main resource and focus on mining information from the code [10]. It might use some information from other resources but the source code is the main information source, especially when it is the case that the source code is the only reliable information source, such as in legacy systems.

Program comprehension goals can be used to overcome existing gaps between what the original designers had envisioned (considering coherent and structured description of a system) and the actual system (whose structure may have disintegrated over time through the absence of an established software process).

The mental models encode the state of understanding of a target program. They are constantly updated in the course of comprehension. Mental models can be built using bottom-up [4] and top-down [5] models. They are frequently combined in hybrid or opportunistic approaches [9]. This happens because the programmer can be viewed as an opportunistic processor capable of exploiting both bottom-up and top-down cues as they became available [9].

Schneiderman [4] presents the bottom-up program comprehension approach in three levels: a) low-level comprehension of the goal of each line of code; b) mid-level comprehension of the nature of the algorithms and data; iii) high-level comprehension of overall program functionalities. Its focus is on internal semantic structure. An important cognitive action in Shneiderman's model is called *chunking*. It is claimed that programmers seldom memorize or comprehend the program line-by-line based on syntax. Rather they recognize *chunks* of code which implements some function such as binary search or swap [8]. This model is associated with a bottom-up process of building internal semantic structure from the source code with the help of syntactic and semantic knowledge.

Brooks [5] presents the program comprehension top-down approach as the process of bridging the gap between application domain and the programming domain through the construction of mappings [8]. There may be intermediate domains in between and hence the mappings are in essence a multi-level structure. Based on this assumption, understanding of a program is achieved when one successfully recovered the mapping among the models within different domains. The recovery process is iterative. The main cognitive action is making and verifying hypothesis. It should start from making hypothesis residing in application domain about the purpose of the program. A verifiable hypothesis is usually specific enough that one can use the so called *beacons* from the source code to verify it. It should be mentioned that Brooks' top-down approach is based fundamentally on searching program source code for verification of an existent hypothesis.

Program comprehension is all about building mental models at different abstract levels with the source code lying on the lowest level. A program is rarely understood in terms of just the text. Rather, models are composed of more abstract notions by interpreting the source code. In this context, *chunks* and *beacons* are all more abstract than simple text streams.

Despite the challenges, many benefits are expected from the use of visualization techniques to reach program comprehension goals. One of our hypotheses is that the amount of time to acquire program comprehension using visualization is smaller than that associated with the approaches used in the current Software Integrated Development Environments (IDE). Moreover, the visualization metaphor seems to be appropriate to identify *chunks* and *beacons*.

In this paper, we focus on the use of SourceMiner and the treemap visualization technique to aid the programmer to acquire knowledge about the program, stimulating the use of a new software comprehension methodology.

3 Treemaps and the SourceMiner

The use of visual representations has shown advantages when compared with traditional source code program comprehension approaches. Software representation through visualization helps members of projects to remember and recognize – and new members to discover – how the code functions [1][3].

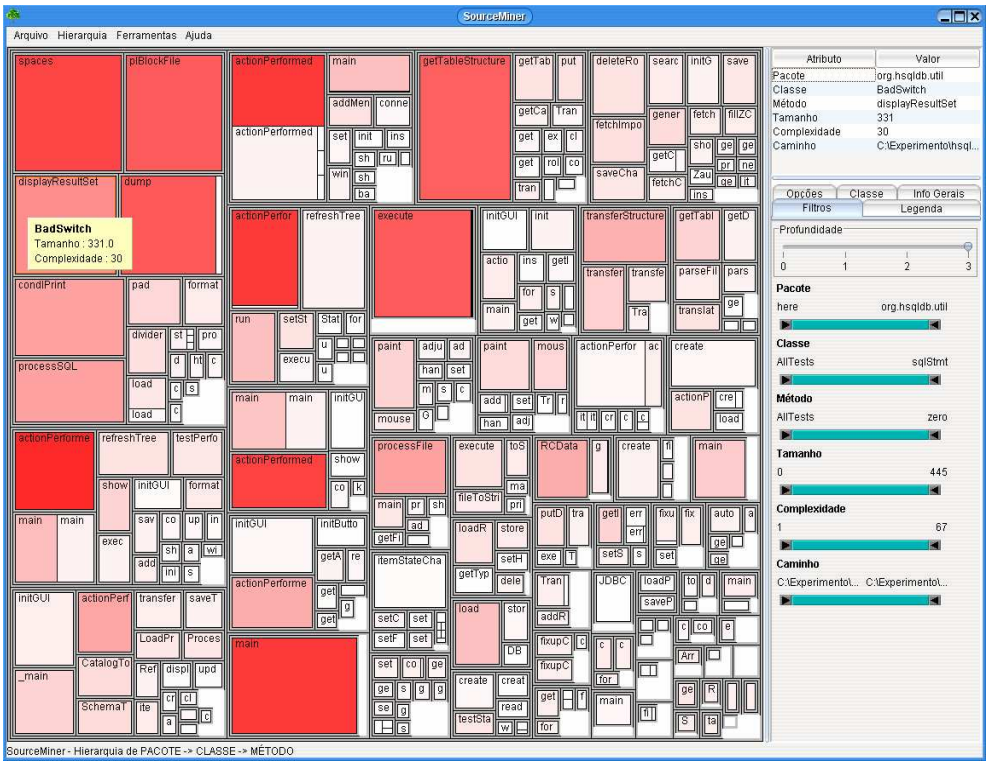


Figure 1 – *SourceMiner* displaying a software source code structure and its size and complexity metrics.

Very large sets of information (even when structurally arranged) are difficult not only to understand, but also to display on a canvas. Based on this premise, visual paradigms must be employed to effectively use the limited size of computer's screens. When analyzing a software program, its source code can contain thousands or even millions of lines. A limited visual paradigm can compromise the program comprehension of a large software system.

Hierarchical visualization paradigms [2] are the best ones to deal with the multiple levels of abstractions associated to the conception of a software system. The most common way to visually represent hierarchical structures is drawing trees. This representation wastes a great portion of the available visual space drawing lines to organize its nodes. As a consequence, big hierarchical structures produce trees that are difficult to visualize.

Treemaps are an alternative method to visualize hierarchical structures [2]. It maps big hierarchical structures as nestled rectangles that fulfill the bi-dimensional space of a canvas. Its main advantage is to show, in an efficient way, huge hierarchies that may contain hundreds of thousand of items [2]. For software source code, treemaps can be used to represent: a) structural descriptive information of which classes are located in which packages, and which methods in which classes; b) quantitative structural information such as how many packages there are in an application, which packages has more classes, and how many methods there are in a class; c) code metrics information such as which are the biggest and smallest packages, classes and methods, and which are the most and the least complex packages, classes and methods of an application.

Code metrics can be efficiently shown in treemaps, because this technique can easily present values by means of the size and color of the rectangles. The visual attribute size is especially useful to represent variables that can be hierarchically decomposed.

With this in mind, our group developed *SourceMiner* – a software visualization tool that uses a software visualization technique called *treemaps* [13] to present structural hierarchy of source code. The tool enables the establishment of relationship between source code structural information (packages, classes, methods) and source code metrics (size and complexity) to a higher level of visualization abstraction. Visual attributes such as color and size of rectangles in the treemap are mapped to program properties like complexity and

lines of code. This mapping between visual and source code attributes can be interactively reassigned. Packages and modules can be filtered based on its metrics and other structural attribute values. All interactions occur through easy to operate widgets and have real time feedback on the canvas. This structure creates an environment that is easy to navigate and can be used interactively according to the program comprehension goals and needs of the user.

Figure 1 presents the *SourceMiner* main screen displaying the structural properties of a 75 thousand lines of code software. The software structural organization (packages, classes, methods) is shown as nested rectangles in the treemap visualization metaphor. Classes that are in a specific package are presented together in the visual representation. In the same way, all the methods declared in a class are presented in the same rectangle area related to its class. The rectangles color is associated with the cyclomatic complexity and its size with the LOC measure for the methods represented at the deepest level of the hierarchy.

The colors, fonts and labels, as well as the border size of the graphical structure can all be defined dynamically by users, maximizing their experience and improving the process of cognition. The association between the software metrics (number of lines and complexity) and the visual attributes (size and color of the areas) can also be dynamically redefined by the users. The interactive controls presented by the tool offers the possibility to use features like zooming and filters. These features can be used to assemble the most appropriate visualization scenario to fit the comprehension goals of a user.

This visualization metaphor offers to *SourceMiner* users a level of visualization abstraction higher than those usually offered by traditional integrated development environments. These conditions can be explored as a powerful environment to program comprehension activities. With this approach one can intuitively answer questions related to:

- Descriptive structural information (e.g.: "what classes are in specific package", and "what methods are in specific class");
- Quantitative structural information (e.g.: how many classes a package has, and which class has the most methods);
- Source code size information (e.g.: "what is the largest package, class and method");
- Source code complexity information (e.g.: "what is the most complex class" and "what are the most and less complex methods declared within each class").

All these questions can be answered using *SourceMiner* without looking at the source code.

In order to assemble a visualization scene, SourceMiner parsers the code to obtain its structure and metrics. The current version of SourceMiner works on Java code and can be invoked from the Eclipse IDE as well as a stand alone tool.

4 The Experiment

In order to evaluate the usefulness of the SourceMiner approach, we have conducted an in vitro controlled experiment. The experiment involved ten students (eight undergrads and two graduates). The students were involved with our research group and were interested in working with empirical software engineering [14]. Students were asked to look at the source code of two large Java software systems and look for bad smells [16] and restructuring opportunities [17] in them.

4.1 Research Questions and Hypotheses

The Goal Question Metric (GQM) approach was used to define the goals for this study. The study hypotheses were derived from the goals stated below.

Goal 1: Analyze the treemap visualization paradigm with the purpose of their evaluation with respect to their effectiveness in identifying descriptive structural information about the programs.

> **Hypothesis 1.1:** individuals applying treemap visualization paradigm perform better than individuals using ad-hoc techniques with respect to effectiveness in identifying descriptive structural information about the programs.

Goal 2: Analyze the treemap visualization paradigm with the purpose of their evaluation with respect to their effectiveness in identifying quantitative structural information about the programs.

> **Hypothesis 2.1:** individuals applying treemap visualization paradigm perform better than individuals using ad-hoc techniques with respect to effectiveness in identifying quantitative structural information about the programs.

Goal 3: Analyze the treemap visualization paradigm with the purpose of their evaluation with respect to

their effectiveness in identifying the size and complexity of relevant elements of programs (packages, classes and methods).

> **Hypothesis 3.1:** individuals applying treemap visualization paradigm perform better than individuals using ad-hoc techniques with respect to effectiveness in identifying of the size and complexity of relevant elements of programs (packages, classes and methods).

Goal 4: Analyze the treemap visualization paradigm with the purpose of their evaluation with respect to their effectiveness in identifying Bad Smells [16] and restructuring [17] opportunities.

> **Hypothesis 4.1:** individuals applying treemap visualization paradigm perform better than individuals using ad-hoc techniques with respect to effectiveness in identifying Bad Smells [16] and restructuring [17] opportunities.

4.2 Dependent and Independent Variables

The experiment treatment is the new approach that we wish to evaluate. In this context, the approach applied to perform program comprehension activities is the treatment variable of this experiment. The experiment manipulates two independent variables (those that influence the application of a treatment and thus the result of an experiment):

1. The **tools available to perform program comprehension activities** (TAPPCA). Students either used *SourceMiner* coupled with *NetBeans* [15] or the IDE alone to identify useful information needed to comprehension tasks.
2. The **background experience** of subjects (EXP). The subjects have different levels of experience, three had industrial experience while the others only have experience in class activities.

The dependent variables (factors expected to change or differ as a result of applying the treatment) examined in the study where:

1. The obtained **individual descriptive information about packages, classes and methods** (IDP, IDC, IDM, respectively);
2. The obtained **individual quantitative information about packages, classes and methods** (IQP, IQC, IQM, respectively);
3. The obtained **individual size information about packages, classes and methods** (ISP, ISC, ISM, respectively);
4. The obtained **individual complexity information about packages, classes and methods** (ICP, ICC, ICM, respectively);
5. The obtained **individual bad smells and restructuring opportunities detected** (IBSOD, IROD, respectively);

4.3 Design

Students were required to work individually. The experiment was conducted in a university lab with all activities been executed in the same day over a period of two hours. These conditions were necessary to measure the ability to obtain the information asked in the questionnaires in a specific amount of time.

4.3.1 Subjects

The participants in the experiment were two graduate and eight undergraduate Computer Science students enrolled in the last year coursework. In addition to lectures, subjects were trained in refactoring, bad smells detection opportunities, design patterns and the treemap software visualization technique used in *SourceMiner* tool. The students were informed that the number of restructuring and bad smells opportunities were not part of any grading criteria. But, the quality of delivered answers and the strategy applied to find useful information to reach comprehension activities goals were very important for this study.

4.3.2 Objects and Materials

The programs used in training and in the experiment were Java Open-Source Systems (OSS). The training application used was *BlackJack*, a game with four classes and 2075 lines of code. The two applications selected to the experiment were *HSQLDB* and *Tyrant*. The first is a relational database application supporting SQL. It is in the fourth version and has 316 classes and 75,495 lines of code. The second is a graphical-based fantasy adventure game. At the tenth version, it has 253 classes and 38,356 lines of code. The experiment started with an explanation of the goals of the study, as well as its process. We then ran a training activity on *SourceMiner* usage with the *BlackJack* application and explained the forms to be filled by the subjects during the experiment.

4.3.3 Instrumentation

The design of the experiment involving all the 10 subjects is shown in Table 1.

Table 01 –The Experimental Design.

Participant	*Tool(s) Used*	
	NetBeans	***NetBeans + SourceMiner***
Student 1	*HSQLDB*	*Tyrant*
Student 2	*Tyrant*	*HSQLDB*
…	…	…
Student 9	*HSQLDB*	*Tyrant*
Student 10	*Tyrant*	*HSQLDB*

4.3.4 Data Collection

All the subjects' workings were recorded on specially prepared data sheets. At the beginning of the experiment, the students completed a background and experience questionnaire. This questionnaire asked the students about their previous programming experience (including Java), professional and academic experience, as well as their experience working with refactoring, bad smells, design patterns, software maintenance and reengineering. The next form, the intermediate form, was used during the experiment to record data associated with descriptive and quantitative structural information, as well as size and complexity of relevant elements of the analyzed program. Another form was used to record code improvement opportunities. It was used to record information about bad smells and restructuring opportunities detected in the analyzed program. At the end of the experiment, the subjects completed a post-study questionnaire to explore the subjects' views on the whole experimental process.

5 Results

This section summarizes and discusses the results obtained during the study and associated with hypotheses presented in section 4.1.

The values of descriptive and quantitative structural information as well as size and complexity of relevant elements of the program presented from figures 2 to 5 were obtained from the intermediate forms filled by all the subjects. The values of bad smells and restructuring opportunities detected presented in figure 6 were obtained from a specific form as stated in Section 4. All the values presented in the figures correspond to the total for each variable measured through the experiment. Figure 2 shows the summarized values for descriptive information about packages, classes and methods (IDP, IDC, IDM, respectively), Figure 3 presents quantitative information about packages, classes and methods (IQP, IQC, IQM, respectively); Figure 4 obtained size information about packages, classes and methods (ISP, ISC, ISM, respectively); Figure 5 complexity information about packages, classes and methods (ICP, ICC, ICM, respectively); and finally Figure 6 bad smells and restructuring opportunities detected (IBSOD, IROD, respectively).

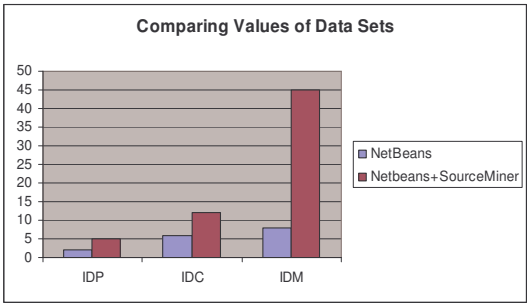


Figure 2 – Comparing Values of Descriptive Information about Packages, Classes and Methods

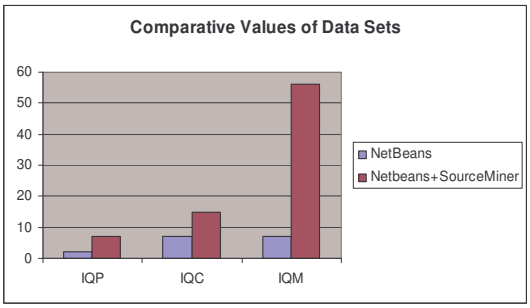


Figure 3 – Comparing Values of Quantitative Information about Packages, Classes and Methods

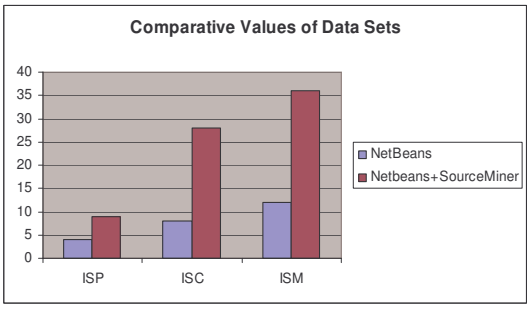


Figure 4 – Comparing Values of Size Information about Packages, Classes and Methods

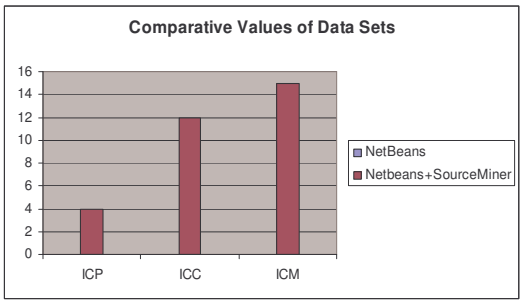


Figure 5 – Comparing Values of Complexity Information about Packages, Classes and Methods

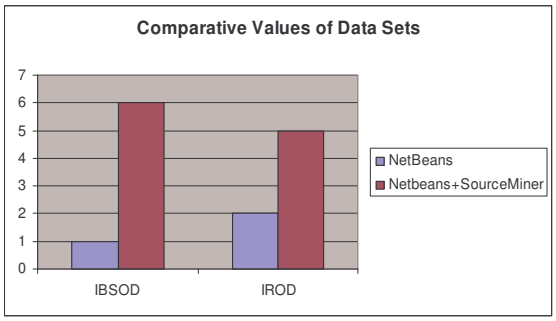


Figure 6 – Comparing Values of Bad Smells and Restructuring Opportunities Detected

5.1 Discussion

Considering Figures 2 to 6 we can conclude that the use of the IDE alone is less effective to obtain information related to the dependent variables described in Section 4.1. As shown in Figures 2 and 3, there is a evidence of the usefulness of program visualization to obtain methods descriptive and quantitative information. The use of IDE alone was not so representative when compared with the IDE plus SourceMiner. Packages and classes descriptive and quantitative information also were greater with SourceMiner but not so representative. These evidences confirm **Hypothesis 1.1 and 2.1.** From size measures registered by the subjects (Figure 4), we can conclude that methods and classes LOC can be more easily obtained using SourceMiner than the IDE alone. From complexity measure there is clear evidence that no subjects were able to obtain it with the IDE alone but with SourceMiner. These confirm **Hypothesis 3.1.** The information registered in the intermediate forms, gave conditions to the subjects to register the detection of Bad Smells [16] and restructuring [17] opportunities in the specific forms. From the results shown in Figure 6, we can conclude that visualization technique was effective to confirm **Hypothesis 4.1** about Bad Smells [16] and restructuring [17] opportunities detection.

The experiment ran without major problems. Some subjects complained about the time window of two hours for program analysis. It might be useful to enlarge this time window. Program comprehension activities are time consuming, and the established time window may have been to short for this purpose. In particular, the amount of bad smells and restructuring opportunities appear to be small (see Figure 6).

5.2 Threats to Validity

The experiment should be replicated in an industrial environment in order to obtain more representative empirical evidence on the bad smells and restructuring detection rates. But as stated in Subsection 5.1, this aspect did not make any influence in the results as well as some of the subjects also have industrial experience. It should be mentioned that all the subjects analyzed the first application with NetBeans only and the second application with NetBeans plus SoureMiner as shown in Table 1. This situation is important to avoid any kind of learning effect in the experiment. The analysis time of the applications *HSQLDB* and *Tyrant* should have been determined in the experiment design. But this did not influenced the results because the initial and final time of each analysis phase (IDE alone and IDE plus SourceMiner) registered by all the subjects indicated a balance in the time elapsed for each phase.

6 Conclusion

Based on the results presented and discussed in Section 5, software visualization techniques are useful to help program comprehension activities.

Based on the hierarchical nature of software systems, results presented in Section 5 shows advantages of using treemap to fulfill program comprehension needs: a) intuitive recognition of software structure according to the packages arrangement of modules; b) easy identification of software modules properties, such as its size and complexity, through the possibility of associating visual attributes with software characteristics; c) complete view of the entire software structure, even in systems with a large number of modules and many attributes; d) easy access to the whole software structure, with the possibility of getting information on demand about a specific part of that structure.

This is the first experiment conducted by the authors to empirically evaluate the assistance of software visualization in program comprehension activities. More experiments have been planned in order to evaluate software product with different levels of size and complexity as well as the experience of the groups that evaluate the software. *SourceMiner* is a tool that combines a visualization paradigm, like treemap, with smart and flexible controls providing a powerful way not only to visualize information, but also to miner it and permit the users manipulate only the necessary information at any moment through the application of data filters. Drawbacks with the treemap have also been identified in the context of the experiment like the inefficiency to detect both the use and absent of design patterns, the coupling and cohesion degree of

software modules, as well as the execution flow of systems. The authors are evaluating how to overcome these limitations in a new *SourceMiner* version with new metrics besides size and complexity already implemented.

The authors have the goal to build an environment where a set of visualization techniques can be selected to fulfill program comprehension activities requirements.

7 References

[1] Fyock, Daniel E. **Using Visualization to Maintain Large Computer Systems.** IEEE Computer Graphics and Applications, p. 73-75. July/August 1997.

[2] Shneiderman, B. **Tree Visualization with Tree-Maps: 2-D Space-Filling Approach**. ACM Transactions on Graphics, v. 11, n. 1, p.92-99, Jan. 1992.

[3] Ng Darren, David R. Kaeli, Sergei Kojarski, and David H. Lorenz. **Program comprehension using aspects**. In *ICSE 2004 Workshop on Directions in Software Engineering Environments (WoDiSEE'2004)*, Boston, MA 02115, May 2004.

[4] Shneiderman, B. **Software Psychology: Human Factors in Computer and Information Systems**. Ablex Publishing Corporation, 1993.

[5] R. Brooks. **Towards a theory of comprehension of computer programs**. Int. J. Man-Machine Studies, 18:543–554, 1983.

[6] A. von Meyrhauser and A.M.Vans. **Program understanding - a survery**. Technical Report CS-94-120, Colorado State University, August 1994.

[7] Scott R. Tilley and Dennis B. Smith. **Comming attractions in program understanding.** Technical Report CMU/SEI-96-TR-019, CMU, December 1996.

[8] Zhang X, Young, M. .**Analysis techniques for Program Comprehension.** Technical Report. Computer and Information Science Department University of Oregon. April 22nd, 2005

[9] S. Letovsky. **Cognitive Processes in Program Comprehension. In Empirical Studies of Programs**. Chapter 5, pages 58-79. Ablex Publishing Corporation, New Jersey, 1986.

[10] Rugaber, S. **Program Comprehension**. Georgia Institute of Technology May 4, 1995.

[11] V. Basili, L. Briand, S. Condon, Y. Kim, W. Melo, and J. Valett, **Understanding and Predicting the Process of Software Maintenance Releases,** 18th International Conference on Software Eng., (ICSE'18), Berlin, Germany, March 25-29, 1996.

[12] Bederson, B.B., Shneiderman, B., and Wattenberg, M. **Ordered and Quantum Treemaps: Making Effective Use of 2D Space to Display Hierarchies** *ACM Transactions on Graphics (TOG)*, 21, (4), October 2002, 833-854.

[13] Shneiderman, B. **Treemaps for space-constrained visualization of hierarchies.** http://www.cs.umd.edu/hcil/treemap-history/. Last updated April 28th, 2006. Accessed in October 2, 2006.

[14] Basili, V. **Experimentation in Software Engineering: Reading Studies**. A Quantitative Approach to Software Management and Engineering - CMSC 735. Experimental Software Engineering Group Institute for Advanced Computer Studies Department of Computer Science University of Maryland and Fraunhofer Center for Experimental Software Engineering – Maryland. 1996.

[15] The NetBeans IDE. Available [ONLINE] from [http://www.netbeans.org].

[16] Fowler, M. Refactoring: Improving the Design of Existing Code. Addison Wesley, 1999.

[17] Mens, T., Touwé, T. A Survey of Software Refactoring. IEEE Transactions on Software Engineering, Vol. 30, No. 2, February 2004.

Un método de Evaluación Ágil del Proceso Software: Agile SPI - Process Assessment Method

Julio Ariel Hurtado Alegría[1,2], César Pardo[1], Luis Fernández[1], Juan Carlos Vidal[1]
[1]Grupo IDIS, Universidad del Cauca, Colombia
[2]Departamento de Ciencias de la Computación, Universidad de Chile, Chile
{ahurtado, cpardo, lefernandez, jcvidal}@unicauca.edu.co

Abstract

La mejora del proceso de software, es una de las estrategias prioritarias para alcanzar una industria de software competitiva. Una de las etapas obligatorias en cualquier programa de mejora es la evaluación del proceso software. Sin embargo, hacer una evaluación completa y profunda del proceso software puede caer en el problema de parálisis del análisis y es uno de los factores de riesgo para el fracaso de un Programa SPI, en especial en el contexto PyME de Software – Pequeña y Mediana Empresa de Software. Este artículo aborda un enfoque basado en principios y prácticas ágiles para la definición de un método ágil de evaluación teniendo en cuenta los requisitos de ARC del SEI e ISO 15504 parte 3. Este método está inmerso en un proceso de mejora ágil: Agile SPI Process con el fin de guiar la evaluación de procesos en el contexto PyME de Software. La aplicación del método en tres casos de estudio arroja un conjunto de lecciones aprendidas que han sido utilizadas para revisar y refinar el modelo.

1. Introducción

Para producir y mantener software de calidad, a bajo costo y en forma efectiva, son de fundamental importancia adecuar los procesos utilizados en su desarrollo y mantenimiento a los objetivos y necesidades de la organización desarrolladora. Un programa de mejora de procesos es un proyecto continuo que conduce a la adecuación de los procesos como estrategia para conseguir las metas del negocio.

El mejoramiento de un proceso se refiere al esfuerzo continuo para saber acerca del sistema de causas de problemas y para usar este conocimiento en el cambio y mejora del proceso con el objetivo de reducir su variación, complejidad y mejorar la satisfacción del cliente [1].

Un programa SPI normalmente involucra seleccionar entre otras cosas, un conjunto de modelos tales como:

Un modelo de calidad que corresponde a un modelo descriptivo que brinda unos referentes de las prácticas a ser implementadas por un proceso de desarrollo de software. De esta manera, un modelo de calidad se compone de un modelo de referencia (Cuerpo de conocimiento normalmente definido alrededor de procesos o áreas de proceso, objetivos y prácticas) y un modelo de evaluación. CMMI[2] e ISO/IEC 15504[3] son modelos de calidad. ISO/IEC 1550 es además un estándar internacional.

Un modelo de referencia es un referente para la industria, el cual especifica las prácticas a ser implementadas por las organizaciones para asegurar algún nivel de cumplimiento expresado finalmente en indicadores de madurez o capacidad. CMMI es un modelo de referencia por sí mismo, organizado por áreas de proceso, objetivos y prácticas generales y específicas, estas áreas a su vez pueden ser agrupadas por categorías o por niveles de madurez según su representación. El modelo de referencia de ISO/IEC 15504:2006 es ISO/IEC 12207:2006[4], el cual está organizado por procesos agrupados por categorías, donde cada proceso es definido por propósitos del proceso y por sus salidas.

Un modelo de evaluación permite facilitar la evaluación de un proceso de desarrollo respecto al modelo de referencia, este se compone a su vez en un framework de medida (estructura de medida del proceso) y su guía de aplicación en un método o proceso de evaluación. CMMI incluye su modelo de evaluación soportado en un framework de Medida basado en dos representaciones y en niveles de capacidad y madurez de los procesos y un modelo de evaluación basado en el cumplimiento de prácticas

generales y específicas a cada área de proceso. En el caso de ISO/IEC el framework de medida está basado en dos dimensiones: dimensión de la capacidad del proceso y dimensión del cumplimiento del proceso. Este cumplimiento también está basado en el cumplimiento de prácticas asociadas al proceso.

Un proceso de mejoramiento es una guía a las organizaciones a adelantar las actividades técnicas y de gestión asociadas a todo el esfuerzo de mejora. Por ejemplo IDEAL [7] es un modelo de mejoramiento asociado a los modelos CMMs del SEI – Software Engineering Institute.

En método de evaluación presenta una guía metodológica para realizar las actividades de evaluación en el marco de un proyecto de mejora o de certificación de un proceso software. Por ejemplo SCAMPI [8] es un método de evaluación asociado a CMMI.

Uno de los problemas para las pequeñas y medianas empresas PyMES, consiste en cómo aprovechar todo el conocimiento adquirido, la experiencia ganada (earned value) y la tecnología desarrollada para impactar positivamente el mejoramiento de sus procesos de desarrollo en situación de poco personal, necesidad de procesos más simples y normalmente pocos recursos.

Con el objeto de cubrir estas necesidades en las PyMES de software fue implementado Agile SPI [5], un framework de mejora que incluye todos los modelos, anteriormente nombrados, para soportar un programa de mejora. Los componentes de este framework tienen la capacidad de poderse componer y complementar con componentes de otros modelos debido a que el núcleo, formado por un marco de definición de procesos y un proceso de mejoramiento, ha sido diseñado de manera independiente a los modelos y estándares de calidad del proceso software.

Una de las actividades fundamentales en cualquier proyecto de mejora es la evaluación del proceso. Normalmente la evaluación del proceso involucra gran cantidad de tiempo, talento humano y costo, lo cual puede resultar determinante para la continuidad de un SPI. De tal forma que es necesario buscar estrategias para enfrentar las actividades relacionadas con la evaluación, así como su gestión.

En este documento se presenta Agile SPI - Process Assessment Method, el cual define el método de evaluación ágil independiente del modelo de calidad, creado al interior de Agile SPI Framework. En la sección 2 se presentan los trabajos relacionados estándares de métodos, métodos y enfoques de evaluación en el contexto PyME. En la sección 3 se presenta el método de evaluación encapsulado en una disciplina de Agile SPI Process. En la sección 4 se presentan los casos de estudio iniciales. En la sección 5 se presentan las conclusiones y el trabajo futuro.

2. Trabajos Relacionados

En todo el mundo ha existido una preocupación por la calidad de los procesos de desarrollo de software para mejorar diferentes industrias nacionales, prueba de esto son modelos y estándares internacionales nombrados anteriormente. Sin embargo muchos de los modelos reconocidos resultan difíciles de aplicar en contextos más pequeños, lo que ha conducido a iniciativas enfocadas hacia las PyMES.

En el caso europeo se han realizado investigaciones en donde se realizan ajustes de modelos de mejora como el IDEAL, para ser aplicados a PyMES MESOPYME [11]; otros proyectos en donde ayudan a empresas de algunas regiones europeas en la mejora de sus procesos, y en el desarrollado de sus propias metodologías de mejora adaptando las mundialmente reconocidas a PyMES, son: ESSI [12], PROCESSUS [13], entre otros.

En el caso latinoamericano modelos como "MoProSoft" [9] de México, "MR mps" de Brasil [10] y Agile SPI de Colombia [5]. Algunos de estos modelos, así como los modelos ISO/IEC y los del SEI, incluyen partes que están directamente relacionados con la mejora y evaluación de los procesos.

Una iniciativa integradora nace con el proyecto Competisoft [17] cuyo objetivo es el de incrementar el nivel de competitividad de las PYMES Iberoamericanas productoras de software mediante la creación y difusión de un marco metodológico común que, ajustado a sus necesidades específicas, pueda llegar a ser la base sobre la cual establecer un mecanismo de evaluación y certificación de la industria del software reconocido en toda Ibero América. Una de las actividades del proyecto Competisoft giran en torno al desarrollo de un Método de Evaluación asociado al modelo de calidad a ser desarrollado en el mismo proyecto, el cual será validado, en el marco del proyecto mediante su aplicación controlada, en empresas y organizaciones de diferentes países de la región CYTED. El método presentado aquí será uno de los insumos al modelo de Competisoft.

2.3. Métodos de Evaluación de Procesos en el Contexto PyME y Normatividad Asociada a la Definición de Métodos de Evaluación de Procesos

Varios son los trabajos asociados a la evaluación de procesos en las PyMES, algunos de ellos adaptando

métodos desarrollados para otros propósitos como SCAMPI (para CMMI) y otros como EvalProsoft desarrollados para tal fin. Los modelos y estándares definen las características esenciales de estos métodos para facilitar un proceso de evaluación. A continuación abordamos tres referentes relevantes que han sido tenido en cuenta para desarrollar y comparar el método de evaluación propuesto.

2.3.1. SCAMPI

Appraisal Requirements ARC [8] define los requisitos que debe cumplir un método de evaluación para CMMI. Con el objeto de dar flexibilidad a los métodos. ARC define una estructura de clases que permite su clasificación y caracterización, como se puede visualizar en la Tabla 1.

Características	Clase A	Clase B	Clase C
Cantidad de evidencia recolectada	Alta	Media	Baja
Puntuación	Si	No	No
Recursos necesarios	Altos	Medio	Bajo
Tamaño del equipo	Grande	Mediano	Pequeño
Requisitos del Evaluador	Certificado	Certificado o entrenado y con experiencia	Persona entrenada y con experiencia

Tabla 1. Caracterización de las clases de métodos de evaluación para CMMI.

SCAMPI es un método de evaluación de clase A, de acuerdo a la clasificación establecida en ARC y puede dar soporte a la conducción de evaluaciones basadas en ISO/IEC 15504. SCAMPI incorpora las mejores prácticas del dominio de evaluación, y está basado en las características de anteriores métodos significativos de evaluación desarrollados por el SEI.

Las principales aplicaciones del método son la mejora interna del proceso, apoyo a la selección del suministrador del software y la monitorización del proceso. El método SCAMPI define las fases de: planificación y preparación, realización e informe de resultados.

En la fase de planificación y preparación de la evaluación se analizan los requisitos de la evaluación, se desarrolla un plan, se selecciona y prepara el equipo, se obtiene y analiza la evidencia objetiva inicial y se prepara la recolección de la evidencia objetiva. En la fase de conducción de la evaluación se examina la evidencia objetiva, se verifica y valida, se documenta y se generan los resultados de la evaluación. En la fase de reporte de los resultados se entregan los resultados de la evaluación y se empaquetan y archivan los activos de la evaluación. El modelo del ciclo de vida del proceso que sigue el método es cascada.

Sin embargo, SCAMPI es un método que normalmente resulta costoso, debido principalmente a que consume una gran cantidad de recursos humanos, de entrenamiento y costo. El costo puede ir de $40.000 a $100.000 dólares por evaluación. El alcance de una evaluación típica SCAMPI puede estar sobredimensionada en una PyME. Sin embargo [18] reporta aplicaciones exitosas en un grupo de PyMEs.

2.3.2. ISO/IEC 15504

La norma ISO/IEC 15504: 2006[3], se enmarca bajo el nombre Tecnologías de Información: proceso de evaluación, y esta constituida por cinco partes. La Parte 1 (En preparación) introduce los conceptos y el vocabulario de términos relacionados con evaluación de procesos. La Parte 2 define las bases para evaluar procesos. La Parte 3 proporciona una guía para la interpretación de los requerimientos para la realización de una evaluación. La Parte 4. define una guía para usar en la determinación de la capacidad del proceso y mejora de procesos. La Parte 5 contiene un ejemplo de un modelo de evaluación de proceso que esta basado sobre el modelo de proceso de referencia ISO/IEC 12207:2006[4].

Esta norma define un marco general para la definición de ***modelos de evaluación*** Parte 2 y los requisitos que debe tener un ***método de evaluación*** Parte 3. Esta norma establece los requisitos tanto para modelos de procesos de referencia como para los métodos de evaluación, sin establecer alguno en particular.

2.3.3. EvalProSoft

El Método de Evaluación, EvalProSoft [9], aplica a las organizaciones dedicadas al desarrollo y/o mantenimiento de software. En particular a las que han utilizado como modelo de procesos de referencia a MoProSoft para la implantación de sus procesos. El Método de Evaluación involucra al Organismo Rector y a la organización a evaluar. La organización selecciona a un Evaluador Certificado reconocido por el Organismo Rector (Asociación Mexicana para la Calidad de Software).

El Método de Evaluación usa como modelo de procesos de referencia la Parte 01: Requisitos de procesos basada en MoProSoft. El modelo de capacidades, que se utiliza para calificar el nivel de

capacidad de los procesos, está basado en la Parte 03: Modelo de capacidades de procesos.

El método cumple con la norma ISO/IEC 15504:2006 parte 3 en la que se definen los requisitos de un método de evaluación en el contexto de la norma ISO/IEC 15504.

El método de evaluación EvalProsoft considera la preparación, actividad previa a la evaluación, y las actividades propias de la evaluación tales como la planeación, ejecución, generación y entrega de resultados y el cierre. El modelo del ciclo de vida del proceso que sigue el método es cascada.

El modelo EvalProsoft es desarrollado en el contexto de la norma mexicana EvalProsoft, el cual está limitado a la norma ISO/IEC 15504, sin embargo la estrategia de realización es similar a SCAMPI. A pesar de ser un modelo creado para las PyMES no incluye estrategias propias para este sector, las cuales recaen más sobre el modelo de evaluación (el cual normalmente permite alivianar las áreas de actuación y el cubrimiento) que sobre el mismo método de evaluación.

2.3.4. Otros Métodos e Iniciativas de Evaluación

Grünbacher [19] describe un proceso de evaluación basado en CMM y adaptado a pequeñas empresas de desarrollo de software, utilizando una herramienta de soporte a la evaluación denominada SynQuest [20]. El proceso define una serie de actividades a ser realizadas pero no define roles y responsabilidades, ni productos de trabajo típicos. El proceso está muy ligado a la herramienta usada, al modelo CMM y aplicado en un proyecto de mejora en una empresa de tipo mediano.

Una iniciativa de evaluación fue el ESPINODE[21] en el marco de una la iniciativa de mejora de procesos en PyMES de desarrollo de software en el Centro de Italia. Se ofrecieron evaluaciones de los procesos gratuitas para todas las empresas que se adscribieron al proyecto. Esto permitió recopilar una gran cantidad de datos derivados de estas evaluaciones y elaborar un informe sobre la madurez de los procesos en las PyMES italianas. Sin embargo en dicho proyecto no se definió explícitamente un nuevo método de evaluación.

Una iniciativa de método de evaluación se tiene a MESOPYME [14], que aunque no define un método de evaluación, define una técnica basada cuestionarios en dos estadios para CMMI. El primer estadio está relacionado a las prácticas específicas y es aplicado a los empleados que realizan los procesos respectivos. El segundo estadio está relacionado a las prácticas genéricas y está orientado a los administradores de los procesos respectivos. En [15] se presenta la aplicación de la técnica para la mejora de los procesos asociados a la Ingeniería de Requisitos.

Un método de evaluación para las PyMES es BootCheck desarrollado en el marco del modelo BOOTSTRAP. Este modelo es compatible con la norma ISO/IEC 15504 parte 3 y ha sido aplicado exitosamente a las PyMES, sin embargo no presenta una estrategia que permita distribuir el esfuerzo de evaluación a lo largo del programa de mejora.

En el caso de Brasil, se tiene el modelo mps Br [10] enfocado a las PyMES. Basado en ISO/IEC 12207:2002, CMMI e ISO/IEC 15504:2003, tiene como objetivo principal ser un modelo para la mejora de procesos de software. El proyecto mps Br, desarrolló dos modelos: un Modelo de Referencia para la mejora del proceso del software – "MR mps" y un Modelo de Negocio para la mejora del proceso del software – "MN mps". El "MR mps" comprende diferentes niveles de madurez y un modelo de evaluación. El modelo de evaluación, a partir de indicadores, asigna un nivel de implementación de una práctica relacionada a un área de proceso, sin embargo no define un método de evaluación.

3. Un Método Ágil para la Evaluación del Proceso Software

Un camino hacia la mejora de procesos no necesariamente involucra pensar única y exclusivamente en un modelo de calidad. Prueba de ello se presenta en [22] donde las PyMES realizan procesos de mejora experimentando y adoptando prácticas implementadas a través de activos de proceso disponibles en la literatura y en el mercado, como por ejemplo prácticas ágiles y prácticas del RUP - Rational unified Process. También usan referentes de mejora como líneas de productos de Software orientadas hacia el reuso a gran escala [23]. En muchos casos las estrategias de mejora son ad hoc, como lo refleja un análisis de la industria de las PyMES colombianas [24] y en algunos casos son estrategias down-top aprovechando modelos fáciles de adoptar como el Personal Software Process PSP [25] y Team Software Process TSP [26].

Pensando en esta independencia de modelos, desde procesos y activos de proceso, modelos de calidad y mejoramiento regionales, así como modelos de calidad y mejoramiento internacionales fue creado el Framework Agile SPI. Este framework es el contexto del método expuesto en este documento. Agile SPI Framework incluye dos elementos asociados a la evaluación de procesos:

1. Un modelo de evaluación basado en la norma ISO/IEC 15504.
2. Un método de evaluación independiente del modelo de referencia, pero que cumple con los requisitos de un método de evaluación ARC Clase A, B y C (dependiendo de la fase en que se aplique) y los requisitos de la norma ISO/IEC 15504 en la parte 4.

3.1. El Contexto del Método

Agile SPI es un Framework de mejora desarrollado para el contexto de las PyMES de software en Colombia [5]. Agile SPI es un Framework basado en modelos livianos que soportan un programa de mejoramiento continuo, a través de un proceso de mejora ágil aplicable a una industria dinámica, creativa, innovadora e incierta como lo es la industria del software. Agile SPI incluye un marco de definición de procesos, FrameworkPDS; un modelo de calidad, Agile SPI – Light Quality Model, y un modelo de mejora, Agile SPI - Process [28]. El modelo de calidad, incluye el modelo de evaluación Agile SPI – Light Evaluation Model [27] y el modelo de mejora incluye como una de las disciplinas, la el método de evaluación Agile Assessment Method.

Por lo tanto, el método de evaluación propuesto está inmerso en el proceso de mejora del Framework Agile SPI: Agile SPI – Process [28] como una disciplina. Agile SPI - Process es un proceso ágil para la ejecución de un programa de mejora del proceso software para las PyMES de Software inicialmente desarrollado para fortalecer la industria colombiana.

Este proceso toma los principios del manifiesto ágil [29] y los requisitos para un SPI liviano [1] con el objeto de adecuar una estrategia SPI en un contexto PyME de software. Para su descripción y asociación a los activos de mejora se ha optado por una visualización basada en SPEM – Software Process Engineering Metamodel como perfil UML.

El proceso se ha definido como ágil-disciplinado, iterativo e incremental y basado en casos de mejora. Los casos de mejora son unidades atómicas de mejora en las áreas de proceso seleccionadas y priorizadas con el fin de gestionar el avance del proyecto global de mejora. Estos casos de mejora son priorizados y en mini proyectos que pueden ser ejecutados en serie o en paralelo a través de iteraciones. Esta priorización y agrupación de necesidades de mejora permite estar liberando de manera más frecuente resultados de evaluación y mejoramiento, facilitando visibilidad, retroalimentación, financiación y entusiasmo suficientes para darle continuidad al proyecto de mejora. Parte del ciclo de vida puede ser visualizado en la Figura 1. donde se presenta solamente las fases de diagnóstico, ejecución y mejora (para ver el proceso completo remitirse a [28]) encadenadas a través del concepto de iteración. Paralelo a estas actividades hay actividades de gestión encapsuladas en forma de disciplinas.

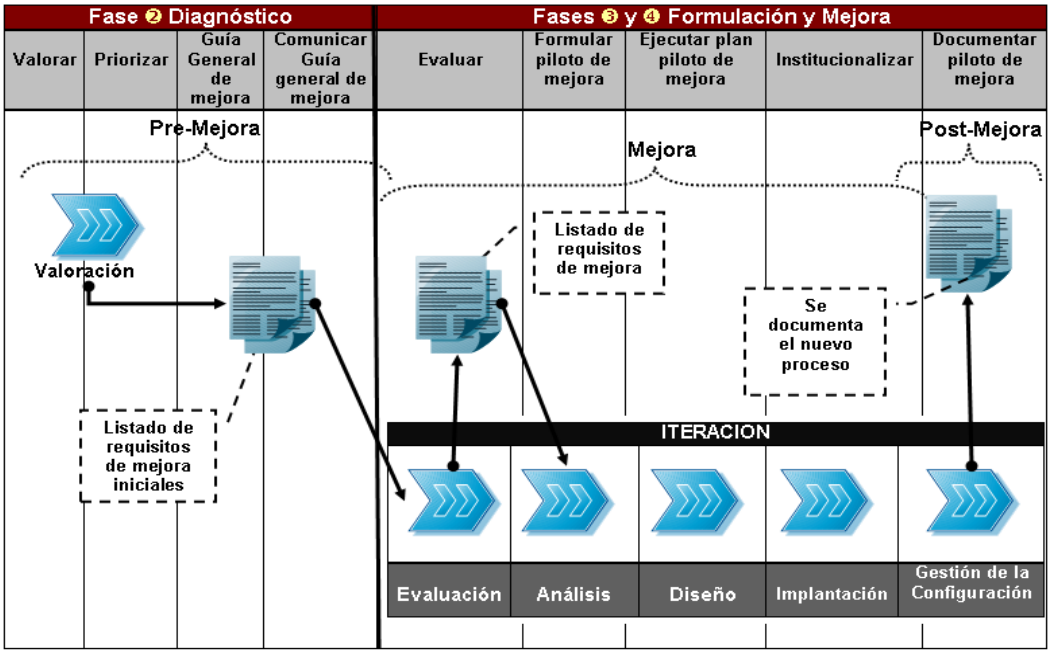


Figura 1. Modelo de Proceso de Agile SPI - Process

Agile SPI - Process está organizado en disciplinas. Una disciplina es un cuerpo de conocimiento y experiencia que está presente en todas las fases e iteraciones en un mayor o menor grado. Las disciplinas presentes en Agile SPI – Process son: entrenamiento, gestión, evaluación, análisis de resultados, diseño de procesos, implantación de procesos, y aprendizaje. El método aquí expuesto corresponde a la disciplina de evaluación.

3.2. Descripción de Agile SPI Assessment Method

Con la evaluación se busca examinar áreas de procesos, el proceso, el estado del proyecto de mejora, el desempeño del programa de mejora, entre otras necesidades. Dependiendo de la fase en la que se esté trabajando y de lo que está evaluando, puede realizarse una evaluación rigurosa o una valoración.

Normalmente, para conocer el estado inicial de todos los procesos basándose en un Modelo de Referencia, en la fase de diagnóstico o de instalación, el realizar una evaluación requiere de demasiado tiempo, así que puede resultar más conveniente realizar sólo una valoración y para cada mini-ciclo de mejora en la fase de mejoramiento realizar una evaluación profunda y focalizada.

Para el desarrollo de la evaluación se requiere el acompañamiento del líder del proceso de la empresa que será evaluado, con el fin de marcar directrices y evaluar el alcance de objetivos generales, así como el evaluador y los demás integrantes del proceso para verificar el cumplimiento de las prácticas correspondientes.

Agile SPI Assesment Method es definido como una disciplina y juega un rol importante en tres momentos de Agile SPI Process: diagnóstico, formulación y mejora, y revisión.

Esta discriminación de momentos de la evaluación permite darle un enfoque más dinámico al proceso brindando agilidad, ahorro de esfuerzo y actualidad de los resultados. Una característica principal del método es que distribuye el esfuerzo de evaluación a lo largo del proceso de mejora.

Esta distribución se logra a través de su aplicación general y superficial en la fase de diagnóstico y por su aplicación específica y profunda en cada una de las áreas involucradas en los casos de mejora a desarrollar en un mini proyecto de mejora dentro de las fases de formulación y ejecución. La actualidad hace referencia a que justo antes de evaluar un área se tiene una evaluación reciente de la misma.

Otra característica principal del método es que se ha mantenido independiente del modelo de calidad, así como el modelo de evaluación, con el fin de no intervenir en las decisiones de la PyME, ya que estos modelos dependen de la decisión tomada al instalar el programa de mejora.

A continuación se describe los elementos del método a través de una disciplina: su estructura estática y dinámica, los participantes, los productos de trabajo y las herramientas.

3.2.1. Componentes de la disciplina

En las Figuras 2 y 3., se puede visualizar los componentes del proceso involucrados. Su estructura estática, dinámica y principales descriptores basados en una disciplina SPEM.

3.2.1.1. Participantes y Responsabilidades

Aunque el diagrama de actividad sólo muestra los líderes de cada actividad, los participantes principales de este proceso son cuatro. El ***evaluador*** tiene la responsabilidad de liderar el proceso de planificación, ejecución y entrega de resultados de evaluación. Un representante del ***grupo de gestión*** definido en el programa SPI, el cual facilita la información necesaria para que la evaluación esté alineada con los objetivos de mejora y tiene la responsabilidad de realizar seguimiento y control de todo el proyecto de mejora, lo cual incluye la evaluación. Los ***líderes de las áreas*** de proceso involucrados en la evaluación, puesto que es sobre sus procesos que se realizarán las actividades lo que involucrará consumo de recursos y alteración de su desempeño normal. Los ***participantes del proceso***, puesto que es a través de ellos que se recogen evidencias específicas sobre las prácticas utilizadas. Todos los participantes tienen la responsabilidad de participar de las actividades de entrenamiento identificadas para a cada uno de ellos.

3.2.1.2. Actividades

Las actividades de la evaluación en Agile SPI Process se describen a continuación.

Definir los objetivos de la evaluación: el evaluador (acompañado de un representante del equipo de gestión y el líder de proceso), en el cuál se establece qué es lo que se quiere evaluar, el proceso completo, un área o un conjunto de áreas de acuerdo al modelo de referencia seleccionado durante la fase de instalación del programa de mejora.

1. **Entradas:** plan de mejora, resultados de evaluaciones anteriores, referente a evaluar.
2. **Salidas:** Objetivos de evaluación establecidos, Requisitos de entrenamiento.

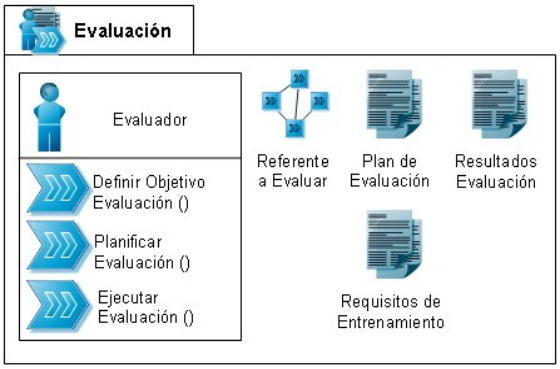


Figura 2. Estructura estática de la disciplina de evaluación

Planificar evaluación: el evaluador desarrolla un plan para el desarrollo de la evaluación. Esto incluye planificar tiempo, recursos humanos y técnicos. En los recursos técnicos se deben definir y diseñar las herramientas necesarias para la recolección de evidencias. Se definen los requisitos de entrenamiento para las personas que estarán involucradas en la evaluación. Una vez esté el plan este deberá ser revisado y aprobado por la gerencia y los líderes de proceso, con el fin de que este quede enmarcado en las condiciones temporales y espaciales de la empresa.

1. **Entradas:** plan de mejora aprobado, objetivos de evaluación, requisitos de entrenamiento.
2. **Salidas:** plan de evaluación aprobado, plan de entrenamiento, herramientas de soporte.

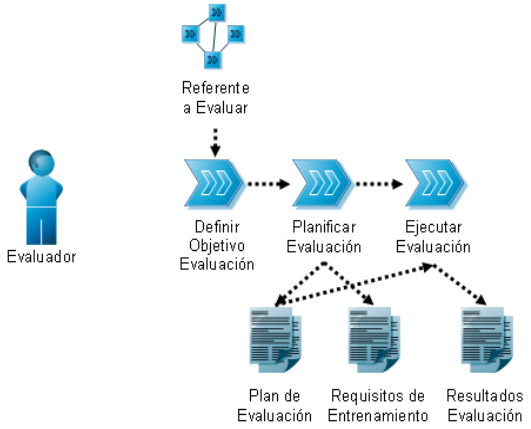


Figura 3. Estructura dinámica de la disciplina de evaluación

Ejecutar evaluación: se lleva a cabo el plan de evaluación establecido. Se ejecuta el plan de entrenamiento definido, se verifica el alcance de los objetivos de entrenamiento, se aplican las técnicas de recolección de evidencias, se obtienen los resultados y se sacan los perfiles de evaluación definidos.

1. **Entradas:** plan de evaluación aprobado, plan de entrenamiento, herramientas de soporte.
2. **Salidas:** Reporte de resultados: perfil de evaluación y evidencias documentadas.

Entrega de resultados: el evaluador entrega los resultados a través de una reunión donde se invita a los líderes de los procesos involucrados, un representante de la gerencia y al líder de la mejora del proyecto de mejora.

1. **Entradas:** Reporte de resultados de evaluación.
2. **Salidas:** Acta de entrega y comentarios relacionados.

3.2.1.3. Productos de Trabajo Principales

Los principales productos de trabajo presentes en esta disciplina son los siguientes:

Plan de evaluación: el plan de evaluación es un documento que incluye los objetivos de la evaluación, actividades y productos de trabajo. Asigna roles y especifica responsabilidades en detalle. Incluye un plan de entrenamiento para organizar las actividades de capacitación. Estable los recursos humanos, tecnológicos y financieros para su realización.

Requisitos de Entrenamiento: es un documento que define las habilidades y conocimientos requeridos para la realización de la evaluación, tanto para el evaluador, así como, para los evaluados.

Reporte de la evaluación: condensa toda la información del resultado de una evaluación. Documenta: hallazgos, evidencias de la implantación de prácticas, niveles de madurez (si se trata de todo el proceso), niveles de capacidad (para el caso de una sola área), perfiles de desempeño, entre otros.

Referente a evaluar: corresponde al modelo de referencia, proceso o indicadores de referencias con los que se pretende contrastar un proceso o un área del proceso para fines de evaluación.

3.2.1.3. Disciplinas Complementarias

El método de evaluación está inmerso en un modelo de mejora Agile SPI- Process. Si ese contexto es mantenido puede aprovechar los roles, actividades y productos de trabajo de disciplinas complementarias a la evaluación las cuales son: gestión de mejora, gestión de la configuración del proceso, aprendizaje y entrenamiento. Cada una de estas disciplinas define

actividades, roles y productos de trabajo transversales al método de evaluación.

3.2.2. Activos Livianos de Proceso Orientados a la Evaluación en Agile SPI - Process

Agile SPI – Process tiene asociados activos de proceso asociados a su mejora. Estos activos incluyen sus mismos componentes (principios, disciplinas, roles, actividades y productos de trabajo), así como plantillas de referencia para documentar los productos de entrada y salida. El proceso busca alivianar la carga dejando los productos de trabajo mínimos, y minimizando el contenido de cada uno de ellos sin perder la objetividad del papel que juegan los documentos en una iniciativa SPI. Estos siguen la premisa: ***"haz lo necesario, si alcanzas, haz algo más"***. Acompañando a Agile SPI - Process hay activos de proceso para **hacer lo necesario.** Particularmente para las actividades de evaluación se cuenta con: guías de entrevistas, encuestas, contratación de personal, plantillas de planificación, reporte de evaluación, matriz de hallazgos, de perfiles de capacidad y madurez para CMMI e ISO/IEC 15504. El método tiene asociado un modelo de evaluación liviano para las PyMES basado en ISO/IEC 15504[27].

Algunos de los activos de proceso, son ágiles y livianos: principios, roles y prácticas adaptados desde métodos ágiles y otros modelos como Scrum para la gestión, TSP y PSP, para guiar el trabajo personal y de los equipos.

El modelo está soportado por herramientas como SPQA.web [30], una solución web del framework Agile SPI, que automatiza la valoración de procesos con referentes CMMI nivel 2 e ISO/IEC 15504, y por Framework PDS [31] que entre otros objetivos, facilita visualizar la arquitectura del proceso, los participantes y sus componentes con fines de verificación de evidencias en la evaluación.

4. Casos de Estudio

Agile SPI - Process ha sido y sigue siendo aplicado en tres empresas pequeñas del suroccidente colombiano: Unisof, Sidem Ltda, Seratic y Cohesión, en donde se tienen evidencias de su aplicación. La Tabla 2. Presenta un resumen de las empresas que aplicaron el método de evaluación, su tamaño empleados total/desarrolllo, el modelo de referencia seleccionado- MR, el modelo de evaluación utilizado ME y los productos y servicios que ofrecen. Uno de los primeros resultados fue haber podido experimentar exitosamente con el método de manera independiente al modelo de referencia y de evaluación utilizados.

La valoración fue llevada a cabo entre integrantes del grupo IDIS y representantes de las mismas empresas. La herramienta de valoración utilizada fue SPQA.web. El grupo IDIS brindó la capacitación y acompañamiento al personal en la aplicación del método de evaluación.

Emp.	Nro Empl.	MR	ME	Producto/ Servicio
Unisoft Ltda	6/3	CMMI Continuo	Light Evaluation Model [27]	Educación: reportes y nómina
Seratic Ltda	6/4	CMMI Niveles	CMMI Nivel 2	Telefonía Celular: Juegos
Sidem Ltda	20/11	CMMI Niveles	CMMI Nivel 2	Software de Administración. Software financiero.

Tabla 2. Empresas del Caso de Estudio

Como resultado de la valoración, se emitió un informe general sobre la situación de todas las áreas contempladas en los modelos de referencia a través del modelo de evaluación asociado. El método de evaluación, los formatos o productos de trabajo definidos agilizaron la gestión y administración de la evaluación. El esfuerzo de las evaluaciones se concentró en un 80% en su diseño (el cuál fue el más arduo), y sólo un 20 % en su aplicación.

Con una evaluación completa de todo proceso de acuerdo a los referentes, el esfuerzo es mucho mayor (5 veces), como ocurrió en el primer proyecto de mejora en la empresa Sitis, donde se hizo una valoración para CMMI nivel 2. En Sitis Ltda. se iniciaron labores de mejora en el marco del proyecto SIMEP-SW cuando el método de evaluación aún no había sido desarrollado en el 2005. Usando un método tradicional de evaluación(basado en SCAMPI) que no prosperó en gran medida debido a la falta de dinamismo del programa de mejora, incluyendo una evaluación que desde un principio abarcaba todas las áreas de proceso definidos por el modelo de evaluación (CMMI Nivel 2). Esto implicó atraso en las actividades de mejora y desmotivación del personal, convirtiéndose en uno de los factores de fracaso del proyecto.

Debido a la rápida respuesta entre evaluación y mejora, brindado por el nuevo método, los resultados mostraron que se ganó confianza sobre el trabajo de mejora y la visualización continua de los cambios de mejora rápidos permitió mantener el interés los programas de mejora. Debido a la disminución del esfuerzo, la retroalimentación continua y la actualidad

de las evaluaciones los nuevos proyectos de mejora han podido seguir exitosamente.

5. Conclusiones y Trabajo Futuro

La evaluación del proceso software es una actividad clave para establecer el estado de los procesos con fines de mejora, su verificación o de certificación. Se ha presentado un método de evaluación independiente del modelo de calidad enfocado a la evaluación interna de una PyME de software basado en las recomendaciones de RAC de CMMI o ISO/IEC 15504 parte 3.

El método de evaluación viene inmerso en la disciplina de evaluación presente en Agile SPI - Process. Para éste se ha definido su contexto, principales componentes y activos de proceso asociados y se ha presentado su aplicación y resultados de la aplicación en tres casos de estudio industriales.

La característica principal del método es la distribución del esfuerzo de evaluación a lo largo del proceso de mejora. Esta distribución se ha logrado a través de su aplicación general y superficial en la fase de diagnóstico y por su aplicación específica en cada una de las áreas involucradas en los casos de mejora a desarrollar en un mini proyecto de mejora. Uno de los principales resultados fue evidenciar las ventajas de esta estrategia en los casos de estudio.

En estos momentos se está desarrollando Agile SPI-Process Manager una herramienta web para la administración y gestión de proyectos o programas de mejoramiento de procesos de software; esta herramienta esta basada en el proceso de mejora Agile SPI – Process, el cual incluye el método de evaluación.

Como trabajo futuro se pretende, a través de los casos de estudio piloto, refinar el método, desarrollar una herramienta de soporte a la gestión del proceso de evaluación, instanciar la disciplina en FrameworkPDS y realizar un proceso de aplicación significativa de todo el Framework Agile SPI con el fin de validar y evolucionarlo junto con todos los componentes.

Aunque hubiese sido ideal manejar un modelo de referencia diferente a CMMI para visualizar mejor la independencia del método de evaluación, la experimentación no fue controlada en ese sentido, debido a que la mejora de las empresas piloto fue dirigida por cada uno de sus objetivos de negocio. Para todos los casos CMMI fue el seleccionado de acuerdo a sus metas organizacionales. A corto plazo se espera aplicar el método en Cohesión, una empresa PyME que sigue el enfoque de Líneas de Producto de Software con el fin de evaluar el cumplimiento de la implementación del framework de Líneas de Producto del SEI [23] y de este modo aplicar el método de evaluación en contexto diferente de mejora.

Con la experimentación también se espera poder madurar los activos de proceso asociados a la evaluación y ponerlos a disposición de la pequeña y mediana industria de software a través del proyecto COMPETISOFT.

Agradecimientos

Este trabajo ha sido realizado en el marco de los Proyectos COMPETISOFT, financiado por el Programa Iberoamericano de Ciencia y Tecnología para el Desarrollo – CYTED; SIMEP-SW, financiado por Colciencias y la Universidad del Cauca, y NIC Chile a través de la Beca NIC Chile 2006 al Departamento de Ciencias de la Computación – Universidad de Chile.

Referencias

[1] Fuggetta, A., Conradi R., Improving Software Process Improvement. 2.002. IEEE Software July/August 2002 (Vol. 19, No. 4).

[2] Software Engineering Institute. CMMI for Systems Engineering, Software Engineering, Integrated Product and Process Development, and Supplier Sourcing (CMMI-SE/SW/IPPD/SS, V1.1) Staged Representation. CMU/SEI-2002-TR-012 ESC-TR-2002-012. 2.002.

[3] ISO/IEC 15504:2006. Information technology – Process assessment –.. 2006. International Organisation for Standardization. 2006.

[4] ISO/IEC. ISO/IEC 12207:2006. Information technology - Process Cycle Life. International Organisation for Standardization. 2006.

[5] Hurtado J. et Al. Agile SPI. Reporte Técnico SIMEPSW-RT-14. 2.005.

[6] ISO/IEC 12207. AMENDMENT 1: Information Technology - Software Life Cycle Processes Amendment 1. 2002.

[7] IDEAL[SM]: A User's Guide for Software Process Improvement. CMU/SEI- 96- HB-001.

[8] Software Engineering Institute. Standard CMMI[SM] Appraisal Method for Process Improvement (SCAMPISM), Version1.1. Method Definition

Document. CMU/SEI-2001-HB-001. 2.001.

[9] Oktaba H. et Al. Evalprosoft. Secretaría de Economía Mexicana. 2.004.

[10] Weber K. Modelo de Referência para Melhoria de Processo de Software: uma abordagem brasileira. Memoerias del XXX CLEI 2004. Arequipa, Perú. 2.004.

[11] Kautz, K., et. al. "Applying and Adjusting a Software Process Improvement Model in Practice: The Use of the IDEAL Model in a Small Software Enterprise". Proceedings of the ICSE, pp. 626-633.

[12] ESSI. *European Software and System Initiative*. http://www.cordis.lu/esprit/src/essi.htm#ch1_1. Último acceso, octubre de 2006.

[13] Horvat, R.V., Rozman, I. and Gyorkos, J. "Managing the Complexity of SPI in Small Companies". *Software Process: Improvement and Practice*, vol. 5, nº 1, marzo 2000, pp. 45-54.

[14] Calvo-Manzano J. Et al. A Software Process Improvement Solution for Small and Medium-Size Enterprises. Proceedings of the First International Research Workshop for Process Improvement in Small Settings. 2.005.

[15] Calvo-Manzano J. et al. Assessment of the requirements management process using a two-stage questionnaire. Proceedings of the Fourth International Conference on Quality Software. IEEE C.S. 2.004.

[16] Oktaba H. et Al. Moprosoft. Secretaría de Economía Mexicana. 2.003.

[17] Proyecto Competisoft. Disponible en http://alarcos.inf-cr.uclm.es/Competisoft/index.aspx . Último acceso octubre de 2006.

[18] García Suz, et. al. SCAMPI A Applied to Small Settings – A Success Story. Presentación. Carnegie Mellon Software Engineering Institute. 2004.

[19] Grunbacher, P. A software assessment process for small software enterprises. Proceedings of the EUROMICRO 97. 1997.

[20] Steinmann C. and Stienen H. Synquest - tool support for software self-assessment. Software Process – Improvement and Practice, 25-12, 1.996.

[21] Mancini, M . A Common Schema for Assessment and Description of Process Improvement Experiments. EUROMICRO 99. 1999.

[22] Coleman, G. An Empirical Study of Software Process in Practice. Proceedings of the Hawaii International Conference on System Sciences. 2.005.

[23] Linda M. Northrop. SEI's Software Product Line Tenets. IEEE Software. July/August 2002. p. 32-40.

[24] Hurtado J. et. Al. Estado de la Práctica del Proceso Software en el Suroccidente Colombiano. SIMEP-SW-RT-19. 2005.

[25] Humprey, W. The Team Software Process. Technical Report CMU/SEI-2000-TR-023. 2.000.

[26] W. S. Humphrey, A Discipline for Software Engineering: Addison Wesley Longman, Inc, 1.995.

[27] Pino F., García F., Piatinni M. Adaptación de las normas ISO/IEC 12207:2002 e ISO/IEC 15504:2.003 para la Evaluación de la Madurez de Procesos Software. 2.005.

[28] Pardo C, et. al. Un proceso Ágil para el Mejoramiento de Procesos de Desarrollo Software en las PyMES: Agile SPI Process. IV Simposio Internacional de Sistemas de Información e Ingeniería del Software en la Sociedad del Conocimiento. Colombia. 2.006.

[29] Manifesto for Agile Software Development. http://www.agilemanifesto.org/ . Último acceso Septiembre de 2.006.

[30] Pino F., Garcia F., Piattini M. Herramienta de Soporte a la Valoración Rápida de Procesos Software. JISBD 2.006. 2006.

[31] Vidal, J. et. al. Hacia un Marco de Trabajo para la Definición de Procesos de Desarrollo de Software: Framework PDS. IV Simposio Internacional de Sistemas de Información e Ingeniería del Software en la Sociedad del Conocimiento. Colombia. 2.005.

[32] BOOTSTRAP Institute: BootCheck assessment tool, http://www.bootstrap-institute.com.

MUM - Proceso de Desarrollo de Software Modularizado, Unificado y Medible

Beatriz Pérez
bperez@fing.edu.uy

Lucía Pedrana
lpedrana@adinet.com.uy

Marcelo Bellini
bellini9@adinet.com.uy

Universidad de la República,
Instituto de Computación,
Grupo de Ingeniería de Software,
Montevideo, Uruguay

Resumen

Se presenta en este artículo las mejoras introducidas en el proceso seguido por los grupos de estudiantes en la asignatura "Proyecto de Ingeniería de Software". Dicho proceso evoluciona año a año, a partir de las mejoras detectadas luego de su puesta en práctica. En el año 2005 se plantea el objetivo de mejorar las mediciones del proceso y mejorar la satisfacción del cliente. Como resultado se obtiene una nueva versión del proceso llamada "MUM - Modularizado, Unificado y Medible". Se presentan los resultados obtenidos de su puesta en práctica y las posibles mejoras en el camino de la mejora continua del proceso de desarrollo utilizado.

1. Introducción

Desde el año 2000, el Grupo de Ingeniería de Software (Gris) de la Facultad de Ingeniería de la Universidad de la República del Uruguay[1] cuenta con un programa construcción y prueba de modelos de procesos que puede ser consultado en [2]. El programa cuenta con tres hitos principales: obtener un proceso definido, obtener un proceso validado y obtener un proceso ajustado y mejorado. La definición de los procesos se realiza por estudiantes del quinto año de la carrera que cursan la asignatura "Proyecto de Grado" durante el primer semestre de clases, obteniendo como resultado el proceso definido, su puesta en práctica se hace durante el segundo semestre en la asignatura "Proyecto de Ingeniería de Software" [3] la cual tiene criterios establecidos que se describen a continuación: la duración del proyecto es de 14 semanas, los grupos de estudiantes son de entre 8 y 13 personas y la carga horaria prevista por estudiante es de 15 horas semanales. Los docentes del curso son los encargados de dirigir a los grupos cumpliendo el rol de "director", los grupos tienen una agenda definida donde se pautan para cada semana, las principales actividades a realizar y los productos que se espera se entreguen al cliente y al director. Cada grupo construye un producto para un cliente real. El producto se instala en el ambiente del cliente en las últimas dos semanas del proyecto y no se realiza mantenimiento del mismo.

Al terminar el segundo semestre, se alcanza el segundo hito: obtener un proceso validado. Luego se identifican mejoras al proceso definido, obteniendo un proceso ajustado y mejorado, el cual puede ser puesto en práctica al siguiente año.

Desde el año 2000, se han creado distintas versiones de procesos que son puestos en la práctica cada año. La evolución y descripción de dichos procesos puede ser consultada en [4]. En el camino de la mejora continua, los procesos son mejorados año a año a partir de su puesta en práctica, creando nuevas versiones de los mismos. En el año 2004 se mejoran los procesos anteriores creando una nueva versión llamada Factor que tiene un esqueleto común de actividades, entregables y roles, e instancias particulares para desarrollo con Genexus (extensión GX) o para desarrollo orientado a objetos (extensión OO).

Dentro de las mejoras identificadas en el año 2004, luego de la puesta en práctica del proceso Factor se identificaron mejorar las mediciones del proceso y evaluar la satisfacción del Cliente.

La mejora del proceso tiene como objetivo enriquecer la predicción y la calidad de lo entregado, mientras se mantiene (o se mejora) la productividad [5].

La medición ayuda a contestar distintas preguntas asociadas con cualquier proceso del software. Mejora la planificación del proyecto, permitiendo determinar las fortalezas y debilidades de los procesos y productos. La medición también ayuda, durante el transcurso de un proyecto, a determinar su progreso, tomar acciones correctivas y evaluar el impacto de tal acción. La medición, para ser efectiva debe: enfocarse en objetivos, aplicarse a todos los productos, procesos y recursos del ciclo de vida, interpretarse basándose en la caracterización y conocimiento de la organización [6].

Las mediciones permiten cuantificar tanto el proceso como el producto. Proporcionan la visión del desempeño del proceso permitiendo: desarrollar perfiles de los datos de los proyectos anteriores que se pueden utilizar para la planificación y mejora del proceso; analizar un proceso para determinar como mejorarlo; determinar la eficacia de modificaciones en el proceso [5].

En el año 2005 se plantea el objetivo de mejorar las mediciones del proceso y mejorar la satisfacción del cliente. En este artículo se presentan las mejoras introducidas en el proceso con el fin de mejorar las mediciones del mismo y la satisfacción del Cliente. Como resultado se obtiene una nueva versión del proceso llamada "MUM - Modularizado, Unificado y Medible" Se presentan los resultados obtenidos de su puesta en práctica en el año 2005 y las posibles mejoras.

En la sección 2 se describe el proceso Factor utilizado en el año 2004, en la sección 3 se presentan las mejoras introducidas al proceso, en la sección 4 se describen los resultados obtenidos y en la sección 5 se presentan las conclusiones.

2. Proceso Factor 2004

El modelo de proceso de Factor y Extensiones fue desarrollado en el año 2004 [7]. Este proceso hace una factorización de los procesos "Modelo Java" y MoDsGX" de años anteriores. El primero desarrollado para lenguajes orientados a objetos y el segundo para desarrollo con Genexus [8]. La factorización consiste en construir un modelo de proceso abstracto en el que se especifiquen los roles, las actividades y los entregables en forma independiente de las tecnologías de desarrollo. Las Extensiones permiten agregar elementos al proceso factorizado para obtener un proceso de desarrollo para un lenguaje o tecnología específica. La extensión de Factor para desarrollo orientado a objetos se llama "Extensión OO" y para desarrollo con Genexus se llama "Extensión GX". Podrían crearse nuevas extensiones que adapten el proceso a otras tecnologías. En la Figura 1 se muestra la relación de herencia entre Factor y sus extensiones.

Tanto los procesos "Modelo Java" y "ModsGX" como su factorización son una adaptación del Unified Process [9], actualmente conocido como IBM Rational Unified Process [10]. El desarrollo se realiza de manera iterativa e incremental, permitiendo que los riesgos del proyecto sean identificados en cada etapa del desarrollo, ayudando a reducirlos significativamente. El desarrollo está basado en Casos de Uso para capturar los requerimientos y asegurar que éstos guían el diseño, el desarrollo y la verificación del software. Es centrado en la Arquitectura del Software, realizando la definición y construcción de ésta en etapas tempranas del desarrollo, lo que permite reducir los riesgos tecnológicos asociados, y sobre la cual se construirán incrementalmente el resto de las funcionalidades del sistema. Para el modelado en las disciplinas se utiliza el lenguaje UML.

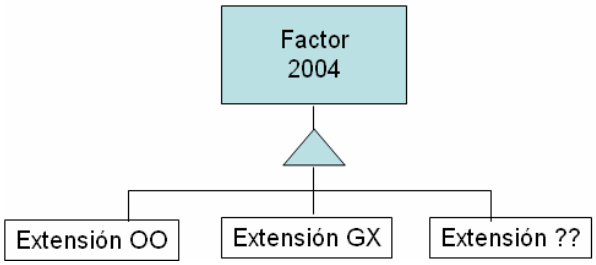


Figura 1– Estructura de Factor

La estructura del esqueleto común de Factor se describe en dos dimensiones, siguiendo la definición de UP: la dimensión del tiempo que muestra los aspectos dinámicos del proceso y se representa mediante ciclos, fases, iteraciones y objetivos, y la dimensión del modelo que muestra los aspectos estáticos del proceso y define Disciplinas con sus actividades y entregables (artefactos), y roles.

Figura 2– Duración de las Fases e Iteraciones

Para adaptar el modelo al curso se define una Agenda en la cual se indica la duración de cada fase e iteración en semanas, de acuerdo a la duración total del curso, indicando para cada semana las actividades del proceso que se deben realizar y los entregables a generar. La duración del proyecto es una variable fija que no se puede modificar, siendo de 14 semanas. En

la Figura 2 se muestra la duración de las fases e iteraciones.

La dimensión del modelo se basa en cuatro elementos de modelado. Estos elementos son las Disciplinas que describen cuando, los roles que describen quién, las actividades que describen como y los entregables que describen qué. Cada Disciplina representa una división de Actividades, las cuales son agrupadas en forma lógica. Para cada Disciplina se describe la secuencia de actividades y los roles que interactúan para obtener los distintos Entregables. Las Disciplinas están divididas en: Requerimientos, Análisis, Diseño, Implementación, Verificación, Gestión de Configuración (SCM), Gestión de Calidad (SQA) y Gestión del Proyecto (GP) [4].

Para el seguimiento y evaluación de los procesos se realizan actividades por parte de los docentes del Grupo de Ingeniería de Software (GrIS): la revisión semanal con el Director, la reunión de Coordinación, las Auditorias y los Cuestionarios de Evaluación. A continuación se describe brevemente cada una de estas actividades, las cuales pueden ser consultadas en detalle en [4].

Semanalmente se realiza una Revisión con el Director del Proyecto donde se realiza el monitoreo de los Entregables más importantes según la Fase e iteración, se discute el estado del proyecto y se toman decisiones en caso de existir desviaciones. El curso tiene además, una reunión de coordinación semanal entre todos los Directores de Proyecto y el responsable del curso, en la cual cada Director presenta un resumen de la revisión con cada uno de sus grupos y de su visión del estado del proyecto en cada caso, poniendo en común problemas para resolver entre todos los docentes y experiencias a tener en cuenta. Esto permite además nivelar el trabajo de los distintos grupos.

Se realizan también auditorias por docentes del curso y por los estudiantes que definieron el proceso. Para la realización de las Auditorias, se identifican las Disciplinas que a ser auditadas en cada Fase y/o iteración. Luego de definidas las Disciplinas que serán auditadas, se identifican los roles involucrados y se convoca a la Auditoria mediante un documento en el cual se detalla la temática y las preguntas a realizar. Posteriormente a las Auditorias se realiza un informe de la reunión el cual se envía a los participantes para que confirmen su acuerdo con la información incluida, luego se envía también a los Directores de Proyecto para que tengan conocimiento de la visión externa de sus grupos. En los distintos años se ha constatado que las Auditorias sirven a los grupos como clases de consulta donde se les explica la razón de hacer determinadas actividades y entregables.

Al finalizar el proyecto se envían cuestionarios finales a estudiantes, directores y clientes, en los que se realizan preguntas sobre el modelo de proceso y desarrollo del proyecto por parte del grupo, y cuyas respuestas son procesadas y utilizadas para ajustar el modelo para la siguiente edición del curso [4].

3. Mejoras introducidas

El modelo de proceso "Modularizado, Unificado y Medible (M.U.M.)" [11] incorpora los ajustes y mejoras que se encontraron luego de aplicar el proceso Factor en el año 2004. En esta sección se describirán las mejoras introducidas en el año 2005, que tienen que ver con la incorporación de nuevas métricas y el seguimiento de la satisfacción del cliente.

3.1 Estudio de Satisfacción del Cliente

Para mejorar el seguimiento de la satisfacción del Cliente, se incorporaron entrevistas previas a terminar el proyecto. Esto, se suma a las encuestas que se realizan al finalizar el proyecto por los docentes del Gris. Las entrevistas se realizaron antes de finalizar la fase de transición del proceso con todos los clientes. Cada entrevista tuvo una duración de una hora y media. Previo a las entrevistas se confeccionó una lista de 30 preguntas, que fue utilizada en el momento de realizar la entrevista. Las preguntas cubren la evaluación de las distintas etapas del proceso desde el punto de vista del cliente así como la conformidad con el producto entregado.

3.2 Métricas

Se incorporaron métricas, tomando como referente los aspectos de calidad interna, calidad externa y calidad de uso propuesto en el modelo de calidad ISO/IEC 9126 [12] para medir un producto de software.

El modelo de calidad ISO/IEC 9126 describe un modelo de calidad para un producto de software en dos partes: a) calidad interna y calidad externa, y b) calidad en el uso. La calidad de proceso contribuye a mejorar la calidad del producto, y la calidad del producto contribuye a mejorar la calidad en el uso. Por lo tanto, evaluar y mejorar el proceso es una forma de mejorar la calidad del producto, y evaluar y mejorar la calidad del producto es una forma de mejorar calidad en uso. De la misma manera, evaluar la calidad en el uso ayuda a mejorar el producto, y la evaluación del producto ayuda a mejorar el proceso [12].

Para medir los aspectos de calidad interna se incorporo la métrica de "Aceptación de los

Requerimientos", que se detalla en la Tabla 1 tomando la definición de los requerimientos en la fase inicial.

Nombre	**Aceptación de los Requerimientos**
Propósito de la Métrica	Mide la satisfacción del cliente, para evaluar la calidad del relevamiento
Fórmula	Aceptación=(A/B)*100
Elementos que la componen	Donde A es la cantidad de requerimientos aceptados por el cliente y B es el número total de requerimientos relevados.
Cuándo se debe registrar	Se efectúa en la fase inicial mientras se relevan los requerimientos con el cliente y se realiza la aceptación de los mismos.
Quién debe registrarla	Responsable de SQA
Dónde debe registrarse	Documento de Validación al Cliente.

Tabla 1– Aceptación de Requerimientos

Para medir la calidad externa, se incorporaron a las métricas "Pruebas cubiertas", "Desempeño de las pruebas" y "Efectividad de las pruebas" en la fase de elaboración y construcción, específicamente en las actividades de verificación, aquí lo que importa son el conjunto de características y atributos que influencian al producto externamente en un entorno de prueba. Las mismas se detallan en las Tabla 2, 3 y 4.

Nombre	**Pruebas cubiertas**
Propósito de la Métrica	Mide qué porcentaje de las pruebas fue efectivamente probado.
Fórmula	Pruebas cubiertas=(A/B)*100
Elementos que la componen	Donde A es el número de pruebas realizadas en la verificación unitaria, del sistema y de integración. Siendo B el número total de pruebas realizadas incluidas las del cliente.
Cuándo se debe registrar	Se efectuará en la fase inicial mientras se realizan las pruebas unitarias, de integración y del sistema.
Quién debe registrarla	Responsable de Verificación
Dónde debe registrarse	Informes de verificación de cada una de las pruebas

Tabla 2– Pruebas Cubiertas

Nombre	**Desempeño de las pruebas**
Propósito de la Métrica	Permiten evaluar qué tanto se corrigieron los errores detectados.
Fórmula	Errores Arreglados=(A/B)*100
Elementos que la componen	Donde A es el número de errores arreglados en la versión N y B el número total de errores encontrados en la versión N-1.
Cuándo se debe registrar	En cada una de las pruebas que se realizan unitarias, del sistema y de integración
Quién debe registrarla	Responsable de Verificación
Dónde debe registrarse	Informe Final de verificación.

Tabla 3- Desempeño de las pruebas

Nombre	**Efectividad de las pruebas**
Propósito de la Métrica	Mide la efectividad para encontrar errores.
Fórmula	Errores Encontrados =(A/B)*100
Elementos que la componen	Donde A es el número de Errores encontrados durante las pruebas y B es Total de errores encontrados: siendo este los encontrados en todas las pruebas realizadas así como también los encontrados por el usuario.
Cuándo se debe registrar	En cada una de las pruebas que se realizan unitarias, del sistema y de integración
Quién debe registrarla	Responsable de Verificación
Dónde debe registrarse	Informe Final de verificación.

Tabla 4 - Efectividad de las pruebas

Tomando como referencia a Roger S.Pressman [13] se incorporaron a las métricas ya existentes la métrica "Productividad orientada al tamaño del producto", que se detalla en la Tabla 5 a los efectos de tener una estimación de la productividad al comienzo en la fase de elaboración y al finalizar el proceso en la fase de transición.

Nombre	**Productividad orientada al tamaño del producto**
Propósito de la Métrica	Se utiliza para tener una estimación de la productividad al comienzo del proyecto, en la fase de elaboración para estimar los puntos de función o líneas de código. Nuevamente al final, en la fase de implantación,

	una vez que se tienen los valores reales. Nos sirve además para comprobar si lo estimado se acercó a lo real
Fórmula	Productividad=A/B
Elementos que la componen	Donde A es la cantidad de Puntos de función o miles de líneas de código y B es el total de horas hombre. Registrándose los valores en el documento de Estimaciones y Mediciones.
Cuándo se debe registrar	En la actividad Estimaciones y Mediciones
Quién debe registrarla	Administrador
Dónde debe registrarse	En el documento de Estimaciones y Mediciones

Tabla 5- Métrica de productividad orientada al tamaño del producto

Por último, para medir los aspectos vinculados a la calidad en el uso incorporaron las métricas de " Estado del funcionamiento 1" y " Estado del funcionamiento 2", para medir la percepción que tienen los usuarios, interactuando con el producto en escenarios específicos de uso, cuando el software esta en la fase de transición. Dichas métricas se muestran en las Tabla 6 y 7.

Nombre	**Estado del funcionamiento 1**
Propósito de la Métrica	Mide el porcentaje de funciones que dio como resultado un dato no esperado (válido) por el cliente.
Fórmula	Datos no válidos=A/B*100
Elementos que la componen	Donde A es el número de funciones con datos inesperados detectadas por el usuario y B es el número total de funciones.
Cuándo se debe registrar	En la actividad correspondiente a las pruebas de Aceptación
Quién debe registrarla	El responsable es el cliente
Dónde debe registrarse	En la documentación correspondiente a las Pruebas Beta (de Aceptación.

Tabla 6– Estado de funcionamiento 1

Nombre	**Estado de funcionamiento 2**
Propósito de la Métrica	Mide el porcentaje de funciones que el usuario encontró dificultad al operar el sistema
Fórmula	Datos no válidos=A/B*100
Elementos que la componen	Donde A es el número de funciones con dificultad detectadas por el usuario y B es el número total de funciones.
Cuándo se debe registrar	En la actividad correspondiente a las pruebas de Aceptación
Quién debe registrarla	El responsable es el cliente

Tabla 7– Estado de funcionamiento 2

4. Resultados obtenidos

En esta sección se describe la experiencia de poner en la práctica en el año 2005 el proceso MUM.

Se describen los resultados obtenidos de la mejora de la satisfacción del cliente y de las métricas introducidas.

4.1 Experiencia 2005

En la edición 2005 del curso Proyecto de Ingeniería de Software, existieron diez grupos de doce estudiantes cada uno para realizar los diferentes proyectos. A cada grupo se le asigno un cliente. Los clientes fueron en su mayoría personas vinculadas al área informática, ingenieros, analistas, docentes de informática o investigadores en el área de informática, existiendo un solo cliente que no está vinculado a esta área específica. En esta edición cada cliente tuvo dos grupos de proyecto lo que le permitiría tener dos experiencias de un mismo proyecto desarrollándolo en diferentes lenguajes.

4.2 Estudio de la satisfacción del Cliente

A continuación se describen los aspectos más relevantes obtenidos en las entrevistas con los clientes.

Las preguntas para evaluar el proceso apuntaban a evaluar las actividades del proceso vinculadas a: requerimientos, gestión del proyecto, verificación y capacitación. La participación de los roles involucrados en el proceso, incluida la del director de proyecto; la calidad de la documentación entrega en las distintas etapas del proceso; la cantidad de horas que tuvo que dedicar el cliente y qué mejoras incorporaría en cada una de las etapas del proceso.

A continuación se describen los principales comentarios obtenidos a partir de las entrevistas con el cliente.

En cuanto al producto se solicito a los clientes que informaran si cumplió mayoritariamente con los requerimientos acordados, su utilidad y conformidad una vez implantado el mismo.

Respecto al proceso, consideran que sería interesante que se realizara una presentación más profunda del mismo y se les entregue un cronograma de las fases e iteraciones junto a los principales objetivos.

Los clientes consideraron que la cantidad de personas que relevaban los requerimientos era la adecuada, que el director del proyecto tiene gran importancia debido a que es quien guía al grupo en eventuales desvíos y ayuda al grupo a definir el alcance del proyecto.

Actualmente las fechas de entrega de los distintos prototipos están dadas por la agenda del proceso. Los clientes consideran que estas fechas deberían acordarse, particularmente en la fase de transición donde se entrega el producto final.

Sobre la documentación que le entregó cada grupo consideran que fue excesiva, lo que algunas veces le dificultaba leerla y aprobarla en forma inmediata debido a que el plazo que tenían no era suficiente. También consideran que debería existir mayor participación de los verificadores en conjunto con el cliente para el armado de las pruebas de verificación y ejecución de las mismas.

En cuanto a la planificación y relevamiento de los requerimientos consideran necesario que previo a las reuniones con el cliente se le envíe los puntos a tratar así como luego de efectuar la entrevista se le envíe el acta de la reunión, aunque esto estaba planificado en el proceso, algún grupo no lo realizaba.

Al terminar el proyecto, se le pidió al cliente que completara una encuesta de satisfacción, la escala utilizada fue del 1 al 5, y podía ser cuantitativa o cualitativa. En la primera 1 es Nada, 2 es Poco, 3 es Medio, 4 es Bastante y 5 es Mucho. En la escala cualitativa 1 es Malo, 2 es Regular, 3 es Aceptable, 4 es Bueno y 5 es Muy bueno.

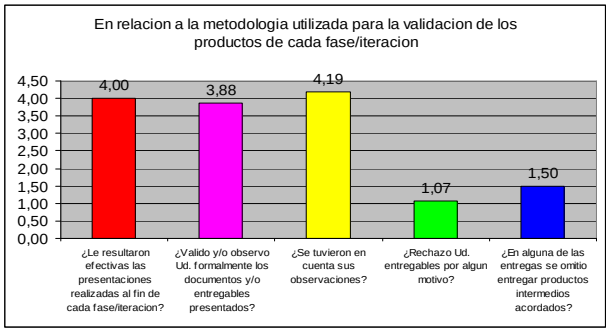


Figura 3– Validación de los Productos

En la Figura 3 puede verse la contestación de los clientes en promedio respecto a la metodología utilizada para la validación de los productos por fase e iteración. En todos los casos las respuestas son ampliamente favorables.

Respecto a la satisfacción con el producto entregado, se puede ver en la Figura 4 que los clientes quedaron satisfechos con el producto final. El cliente de los grupos 6 y 7 no respondió la encuesta de satisfacción.

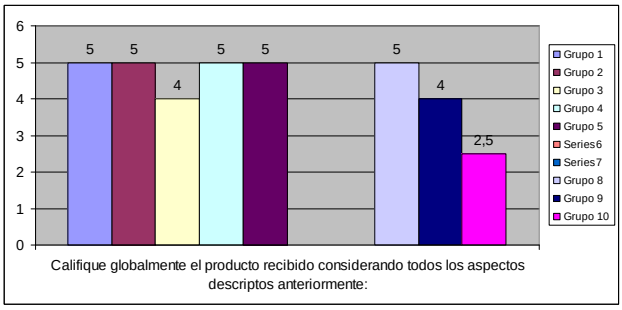


Figura 4– Calificación del Producto

En lo que respecta a la satisfacción respecto al producto y al proyecto en forma global, en una escala del 1 al 12, en la Figura 5 puede verse el resultado para cada grupo.

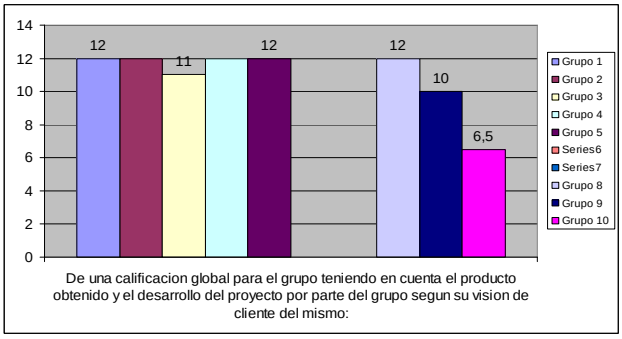


Figura 5– Calificación Global

Se concluye de las entrevistas y las encuestas de satisfacción realizadas que los clientes quedaron muy conformes con el proceso MUM seguido por los grupos y con los productos resultantes de aplicar el mismo. Las sugerencias antes mencionadas para mejorar el proceso y el producto realizadas por los clientes ser{an introducidas en las nuevas ediciones de “Proyecto de Ingeniería de Software”.

4.3 Métricas

Los valores de las métricas son obtenidos por los estudiantes durante el proceso y registrados en los diferentes documentos, que van entregando a lo largo de las diferentes etapas del proceso. A continuación se

detallan los resultados de las principales métricas obtenidas.

Métricas de Aceptación de Requerimientos

Los valores obtenidos están por encima de 0,7. Tomando como referente la norma ISO 9126 las medidas se valoran entre 0 y 1 cuanto más cercano a 1 fue mejor el relevamiento, por lo cual se considera que el relevamiento de los requerimientos fue bueno, el alcance reflejo lo que el cliente quería.

Métricas de Pruebas Cubiertas

El resultado de las métricas de pruebas cubiertas nos da una idea de la cantidad de pruebas que se realizaron de acuerdo a las planificadas y de esta forma tener cierta garantía de que antes de poner en producción el producto se contará con menores probabilidades de errores en la ejecución del mismo.

En este caso todos los valores dieron por encima de la media por lo cual se considera que el producto esta por encima del 50% probado, cuanto más cerca de 1 de cómo resultado la métrica, menor probabilidades de error se obtendrán en producción.

Métricas de Productividad orientada al tamaño

En la Tabla 8 se muestran los valores de la Métrica de productividad orientado al tamaño del producto para cada grupo. Dicha métrica es el cociente entre la cantidad en miles de líneas de código sobre el total de horas del grupo.

Grupo	Métrica de Productividad	Tamaño (LOCs)	Total de Horas
1	**8.24**	40090	4865
2	**9.60**	41371	4310
3	**12.12**	42798	3530.44
4	**9.52**	28848	3029.9
5	**8.83**	38401	4348
6	**2.14**	9374	3471.87
7	**4.87**	18171	3735
8	**2.11**	8722	4138.65

Tabla 8- Métrica de productividad

Se distinguen dos intervalos de valores bien diferenciados, uno donde están los primeros cinco grupos con valores que oscilan entre los 8 y 12 y otro con los últimos 3 (Grupos 6,7 y 8) que obtienen valores entre 2,11 y 4,87. Podemos destacar que el grupo más productivo fue el tres y los menos productivos fueron los grupos seis y ocho.

Métricas de Efectividad de las pruebas

Del total de defectos encontrados, se contabilizan tanto los encontrados en las pruebas realizadas como los encontrados por el usuario en las pruebas de aceptación.

Cuanto más aproximado a uno da el resultado de esta métrica maximizará el número de errores encontrado durante las pruebas. Los resultados obtenidos en los diferentes grupos estuvieron en todos los casos por encima de 0,77 y la mayoría estuvo por encima de 0,85 por lo cual se considera que las pruebas fueron efectivas.

Las resultados de las métricas de estado de funcionamiento 1 y 2 no se pudieron registrar porque el cliente, quien era el encargado de registrarlas, no las entrego.

En general, se puede concluir que las métricas incorporadas nos permiten evaluar distintas actividades del proceso, particularmente las actividades vinculadas a Requerimientos, Verificación, Implementación, Implantación y Gestión del Proyecto. Esta información puede ser utilizada para la comparación con futuros proyectos. Se considera que para lograr una mayor efectividad, se debe hacer un mayor seguimiento por parte de los directores de proyecto para asegurarse que los estudiantes están recabando los valores para el cálculo de las métricas.

Estos valores deben registrarse en los documentos que realizan en cada entrega según corresponda. En el caso de la información necesaria para obtener las métricas correspondientes al "Estado del funcionamiento 1" y "Estado del funcionamiento 2" sería conveniente explicar mejor a los clientes su importancia para que entreguen la documentación de las pruebas operativas y funcionales luego de terminar el curso junto con la entrega de la encuesta de satisfacción al final del curso.

5. Conclusiones

Este trabajo describe las mejoras realizadas sobre el proceso utilizado en "Proyecto de Ingeniería de Software". Las mejoras introducidas apuntan fundamentalmente a la incorporación de mediciones y a la evaluación de la satisfacción del cliente. Se presentan los resultados obtenidos de poner en práctica el proceso MUM que incorpora estas mejoras.

Se concluye que las métricas incorporadas permiten evaluar distintas actividades del proceso particularmente las actividades vinculadas a

Requerimientos, Verificación, Implementación, Implantación y Gestión del Proyecto. Esta información puede ser utilizada para la comparación con futuros proyectos.

Como trabajo a futuro se encuentra mejorar el involucramiento del cliente y de los estudiantes en recolectar los datos necesarios para el calculo de las métricas.

Se evaluó el modelo de proceso M.U.M desde el punto de vista del cliente mediante entrevistas y encuestas realizadas a los mismos. Como consecuencia de haber realizado las entrevistas a todos los clientes y analizando el valor de la información obtenida consideramos que la información obtenida fue de gran valor para conocer la visión del cliente respecto al proceso. Se propone que las entrevistas con el cliente para conocer su visión respecto al proceso seguido, el material entregado, las validaciones y la definición del alcance sean parte del proceso y que se realicen en la semana posterior a la finalización del proceso.

Agradecimientos

El presente trabajo ha sido desarrollado en el marco del proyecto COMPETISOFT (Mejora de Procesos para Fomentar la Competitividad de la Pequeña y Mediana Industria de Software de Iberoamérica) del programa CYTED (Ciencia y Tecnología para el Desarrollo).

6. Referencias

[1] Grupo de Ingeniería de Software (Gris), Instituto de Computación (INCO), Universidad de la República de Uruguay. Sitio Web, 2006. http://www.fing.edu.uy/inco/grupos/gris/

[2] Triñanes J., "TLREQ: Proceso para desarrollo a distancia", CSOC-JIISIC 2003, Valdivia, Chile, 49-52, 2003

[3] Proyecto de Ingeniería de Software Instituto de Computación (INCO), Universidad de la República. Sitio Web, 2006. http://www.fing.edu.uy/inco/cursos/ingsoft/pis/index.htm

[4] Delgado, A. Pérez B., "Modelo de desarrollo de software OO, Experimentación en un curso de Ingeniería de Software",. Proceedings de las V Jornadas Iberoamericanas de Ingeniería de Software e Ingeniería del Conocimiento (JIISIC' 06), ISBN 970-94770-0-5.Puebla, México, 1 al 3 de Febrero de 2006.

[5] Pomeroy-Huff M., Mullaney J, Cannon R., Sebern S. "The Personal Software Process (PSP) Body of Knowledge", Version 1.0., CMU/SEI-2005-SR-003 ,http://www.sei.cmu.edu/publications/documents/05.reports/05sr003.html

[6] Basili V., Caldiera G., Rombach D. "The Goal Question Metric Approach", Wiley& Sons Inc, 1994

[7] Casal Y. "Modelo de Proceso Factorizado". Proyecto de Grado de la Carrera de Ingeniero en Computación, Facultad de Ingeniería, Universidad de la Republica, Uruguay, 2005. http://www.fing.edu.uy/inco/cursos/ingsoft/pis/memoria/experiencia2004/Factor04/index.htm

[8] ARTech, "Visión General de Genexus", http://66.132.154.118/cgi-bin/adcret03.cgi/GeneXus%20Vision%20General.pdf?0,1,688

[9] Jacobson I., Booch G., Rumbaugh J., "The Unified Software Development Process", Addison Wesley, 1999.

[10] IBM Rational Unified Process. http://www-130.ibm.com/developerworks/rational/products/rup/

[11] Pedrana, L. Bellini, M. "Modelo de Proceso Modularizado Unificado y Medible". Proyecto de Grado de la Carrera de Ingeniero en Computación, Facultad de Ingeniería, Universidad de la Republica, Uruguay, 2006. http://www.fing.edu.uy/inco/cursos/ingsoft/pis/memoria/experiencia2005/MUM/index1.htm

[12] ISO/IEC TR 9126:2001. Software Engineering - Product Quality. Part 1 – Quality Model.

[13] Pressman R., "Adaptable Process Model", www.rspa.com/apm

Enfoque de Metamodelado y Multiformalismo Aplicado al Proceso Software usando AToM^3

Mabel del V. Sosa
Departamento de Informática
Facultad de Ciencias Exactas y Tecnologías
Universidad Nacional de Santiago del Estero
Avenida Belgrano (S) 1912,
4200 Santiago del Estero, Argentina.
litasosa@unse.edu.ar

Silvia T. Acuña, Juan de Lara
Departamento de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Avenida Tomás y Valiente 11
28049 Madrid, España
{silvia.acunna , juan.delara}@uam.es

Abstract

En este artículo se presenta el modelado y verificación de los aspectos dinámicos y temporales de un proceso software. Con el metamodelo generado ha sido posible analizar el comportamiento del proceso para determinar la coherencia y consistencia de las especificaciones efectuadas como resultado de los requerimientos del diseño. Esto ha permitido detectar los fallos del modelo y efectuar las correcciones correspondientes. El metamodelo proporciona a los desarrolladores una visión lógica completa del proceso y facilita la selección de opciones de implementación, centrándose en aspectos de proceso y no en un dominio de aplicación en particular. Para el modelado se aplica la herramienta de metamodelado AToM^3, que permite la descripción de sistemas usando múltiples formalismos, así como la traducción y verificación automática de modelos. Se destaca la importancia de contar con este tipo de herramientas como soportes de apoyo a los métodos convencionales de análisis y diseño del proceso software.

1. Introducción

El proceso software tiene por finalidad gestionar, transformar y soportar la necesidad del usuario en un producto de software que satisface esa necesidad [1], siendo el factor crítico la entrega de sistemas software de calidad. El objetivo general del proceso software es mejorar la práctica del desarrollo de software a través de: a) formas mejoradas de diseño organizacional al nivel de procesos individuales y de la organización como un todo; y b) innovaciones complementarias en el soporte tecnológico. En este contexto, el proceso software se define como el conjunto de actividades necesarias para producir un sistema software, ejecutadas por un conjunto de recursos humanos organizados según una estructura organizativa concreta y contando con un soporte de herramientas técnico-conceptuales.

El objetivo último del modelado del proceso software es guiar, controlar y gestionar las actividades del proceso, por tanto, los aspectos estructurales y dinámicos del proceso software deben ser considerados de modo preciso. Este modelado del proceso software requiere de herramientas y métodos formales de especificación que permitan dar soporte al análisis, diseño y comprobación del proceso y, en particular, determinar con precisión la corrección de sus desarrollos [10][12]. Para realizar estas especificaciones, existen diferentes herramientas [4][14][15], sin embargo, se requiere de un formalismo capaz de modelar y representar de forma explícita y gráfica los aspectos de la dinámica del proceso, como así también el análisis y comprobación automática de las propiedades del sistema.

El objetivo de este artículo es presentar un modelo de proceso software usando la herramienta AToM^3 que se basa en el enfoque de metamodelado y multiformalismo [7][9], y proporciona técnicas que pueden aplicarse para mejorar el proceso software. Para modelar los aspectos dinámicos, división de trabajo y responsabilidades, usamos UML. Para completar la formalización, los modelos son traducidos a redes de Petri [13], donde se considera la secuencia y concurrencia de las actividades y orden temporal en que pueden ocurrir los eventos. Finalmente la especificación de las propiedades dinámicas temporales del modelo resultante se realiza mediante un lenguaje de lógica temporal CTL (Computational Tree Logic), y se comprueba mediante técnicas de model checking [3], provistas por el mismo AToM^3.

El artículo se organiza como sigue: en la sección 2 se describe brevemente la herramienta AToM^3. En la sección 3 se detalla la aplicación de AToM^3 al proceso software. En la sección 4 se presentan los trabajos relacionados y finalmente en la sección 5 se exponen las conclusiones del trabajo.

2. AToM^3

AToM^3 (A Tool for Multi-Formalism Modelling and Meta-Modelling) es una herramienta creada en colaboración por la Universidad de McGill en Montreal y la Universidad Autónoma de Madrid [2][7]. La misma es seleccionada porque facilita el modelado de sistemas complejos tal como el proceso software. AToM^3 se basa en conceptos de *metamodelado* y *multiformalismo*, permitiendo la creación, edición, traducción y verificación de metamodelos. AToM^3 permite la descripción de lenguajes visuales específicos de dominio mediante metamodelos. Por ejemplo, se ha usado la herramienta para describir lenguajes visuales tales como diagramas de actividad y otros diagramas UML y redes de Petri, entre otros [8][9].

AToM^3 permite la especificación de transformaciones entre distintos formalismos mediante gramáticas de grafos. Por ejemplo, se han diseñado transformaciones para traducir diagramas de actividad a redes de Petri.

En el ámbito de la verificación de sistemas AToM^3 proporciona una técnica de verificación automática denominada model checking [3], orientada a realizar la verificación mediante la exploración exhaustiva del espacio de estados del modelo en busca de situaciones que no cumplan con las propiedades especificadas. En este último caso la prueba es parcial, pero conservativa, en el sentido de que si el sistema cumple las propiedades, entonces el sistema real también lo hace. Normalmente, en AToM^3 se aplica esta técnica al grafo de cubrimiento o alcanzabilidad derivado de una red de Petri. Por tanto, para el análisis de un sistema mediante model checking, antes se transforma a una red de Petri. Para la especificación formal de los requerimientos a comprobar mediante model checking se utiliza CTL.

3. Aplicación de AToM^3 al Proceso Software

A continuación, se describe la aplicación del multiformalismo que soporta AToM^3 a un modelo de proceso software que sigue una adaptación del proceso de IEEE [6]. Este modelo adaptado agrupa las actividades del proceso software en tres categorías:

- *Grupo de Actividades de Gestión de Proyectos*: Entre estas actividades se encuentran la estimación de tiempos y recursos, planificación y seguimiento y control del proyecto.
- *Grupo de Actividades Orientadas al Desarrollo*: Agrupan las actividades técnicas relacionadas con el desarrollo de software tales como identificación de necesidades, definición de requisitos, diseño, implementación e instalación.
- *Grupo de Actividades de Soporte*: Abarca las actividades que se aplican para complementar tanto las actividades de gestión como las actividades técnicas, entre las que se incluyen evaluación, gestión de configuración, o documentación.

La perspectiva estructural del proceso software está representado mediante diagramas de clases de UML con las clases involucradas en el proceso software de un proyecto específico. Para modelar la perspectiva de comportamiento se parte de los diagramas de clase que representan la estructura del modelo y se especifica mediante diagramas de estados complementados con diagramas de actividades. En la perspectiva temporal se modelan los aspectos de temporalidad cualitativa y se realiza mediante redes de Petri. Para obtener las tres perspectivas mencionadas se usa AToM^3.

3.1. Perspectiva Estructural y de Comportamiento

Con respecto a la perspectiva estructural, en la Figura 1(b) se representa el diagrama de clases con los objetos definidos como jerarquía de clase, los atributos, los métodos y las relaciones generales entre clases. En la Figura 1(d) se muestra un diagrama de objetos que representa la configuración del sistema en forma global. Se representa en el diagrama el objeto Proyecto Software, tres objetos Rol (rol “Jefe de proyecto”, rol “Desarrollador” y rol “Evaluador”) y tres objetos Documentos producidos en alguno de los grupos de actividades de gestión, desarrollo o soporte. Estos diagramas permiten visualizar la estructura del proceso software realizado para un proyecto de desarrollo de software convencional. Un proyecto implica el trabajo de un grupo de personas que conforman el equipo del proyecto, donde cada persona o miembro realiza diferentes actividades involucradas en alguno de los grupos de actividades. En cada grupo de actividad se elaboran documentos como productos de las actividades realizadas. El ejemplo presentado muestra una situación típica del proceso de desarrollo de software representado mediante la Figura 1(b) donde para las dos clases relevantes para el ejemplo que se tratará aquí, rol *Jefe de proyecto* y *Documento*, se han definido los atributos y los métodos.

Con respecto a la perspectiva de comportamiento, la clase *Documento* tiene asociado un diagrama de transición de estados representado en la Figura 1(a). Las clases *Proyecto*, *Rol*, *Equipo del proyecto* no tienen asociado diagrama de transición de estados, pues se asume un diagrama de transición de estados general para todos ellos, compuesto de un solo estado del que todos los métodos de la clase disponibles pueden invocarse. El diagrama de transición de estados para el objeto *Documento*, consta de cinco estados diferentes que representan cuando el documento está siendo creado, revisado, modificado, en proceso y terminado (denominados “En creación”, “En proceso”, “En revisión”, “En modificación” y “Terminado”). El proceso cambia de estado en función a las invocaciones de los métodos *Preparar(), Abrir_Doc(), Modificar(), Revisar(), Guardar_ok() o Borrar()*. En particular, el estado “En proceso” se descompone en un nuevo diagrama de estados que represente las

actividades de alguno de los grupos de actividades de gestión, desarrollo o de soporte. En la Figura 1(c) el estado "En proceso" explota en subestados correspondientes a las actividades del grupo orientadas a la gestión, en donde se encontraría el documento que se completa a medida que se cumple cada actividad del subgrupo de actividades. En este caso, el proceso cambia de estado en función a las invocaciones de los métodos, pertenecientes a la clase "Jefe de Proyecto", (*Recibir_docs(), Iniciar_proc_c(), Iniciar_proc_s(), Iniciar_gestión_c(), Iniciar_gestión_s(), Control(), Cambios(), Enviar_informes ()* y *Finalizar()*).

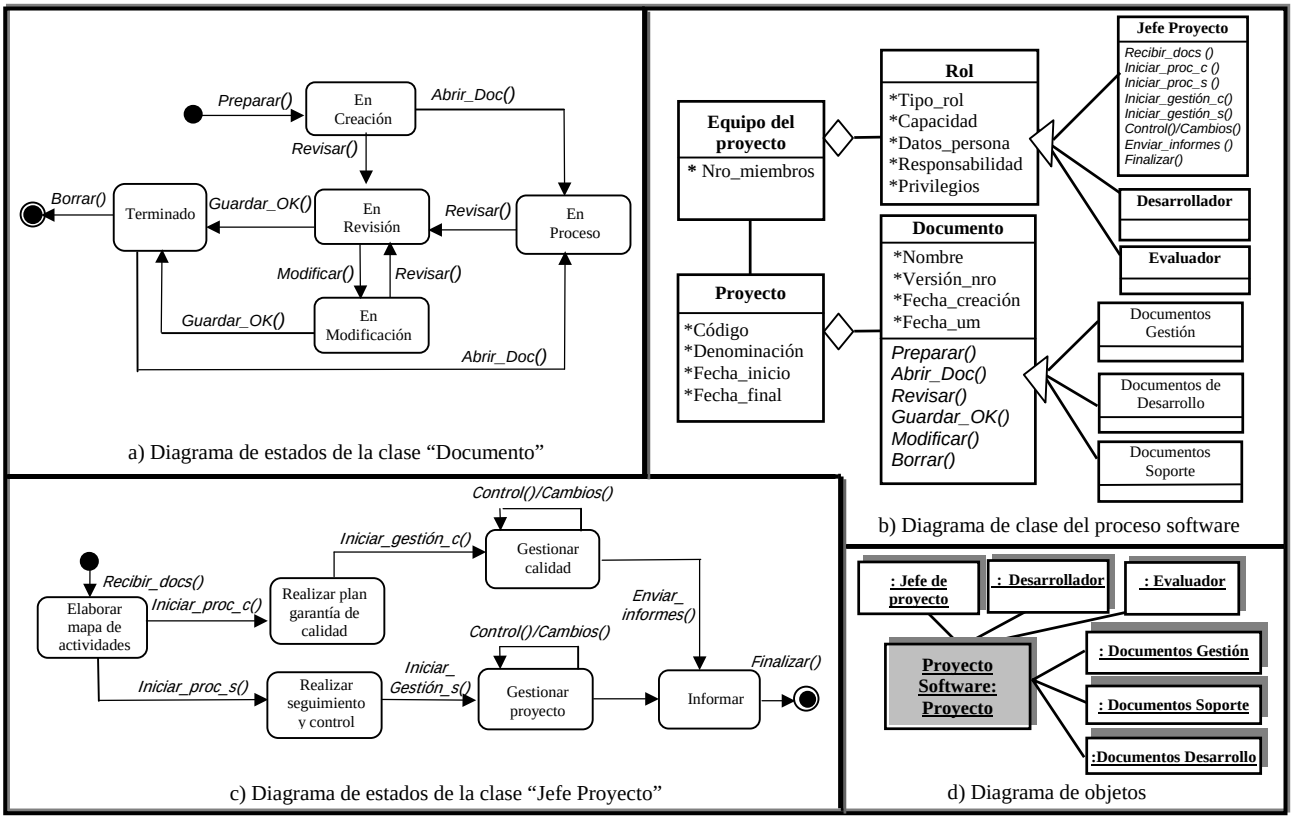


Figura 1. Diagrama de clases, objetos y estados del proceso software

En la Figura 2 se complementa la perspectiva de comportamiento representando el proceso software modelado a través de un diagrama de actividades. El diagrama de actividades está desarrollado en un nivel de abstracción global, en el cual se expresa la concurrencia de las acciones y la división de responsabilidades y roles. Con el diagrama de estados y actividades se logra clarificar y precisar el orden, tanto precedencia como simultaneidad de las actividades llevadas a cabo dentro del proceso software, a un nivel de detalle general.

Mediante las definiciones de las clases, estados y actividades se ha logrado representar la funcionalidad del proceso es sus aspectos esenciales, *estructural*, es decir los componentes que lo integran, sus propiedades y relaciones entre ellos, y *dinámica* expresado en términos de interacción, comunicación y sincronización de dichos componentes. Sin embargo en esta formalización se pretende considerar cuestiones de temporalidad cualitativa explícita para la simulación de características específicas del modelo. Por ejemplo determinar cuándo una actividad debe o puede mediar entre dos actividades cualesquiera del proceso. Para realizar este tipo de especificación se utiliza como herramienta de formalización las redes de Petri según se describe en el apartado siguiente.

3.2. Perspectiva Temporal

Para analizar la exactitud de los modelos construidos, AToM^3 traduce automáticamente el diagrama de actividad a un modelo de red de Petri, para comprobar si el modelo es consistente y verificar las propiedades de la red resultante usando métodos de análisis para redes de Petri. Además, las propiedades pueden ser analizadas generando (implícitamente o explícitamente) un espacio de estados de la red y aplicando model checking [3]. Esta técnica permite verificar si el modelo cumple una cierta propiedad especificada con CTL.

La red de Petri de la Figura 3 representa el proceso software completo de la organización desarrolladora, formado por dos tipos de estados: lugares y transiciones, que en este ejemplo, representan actividades y eventos respectivamente. En la Figura 3 aparecen tres rectángulos que engloban a cada uno de los grupos de actividades por separado.

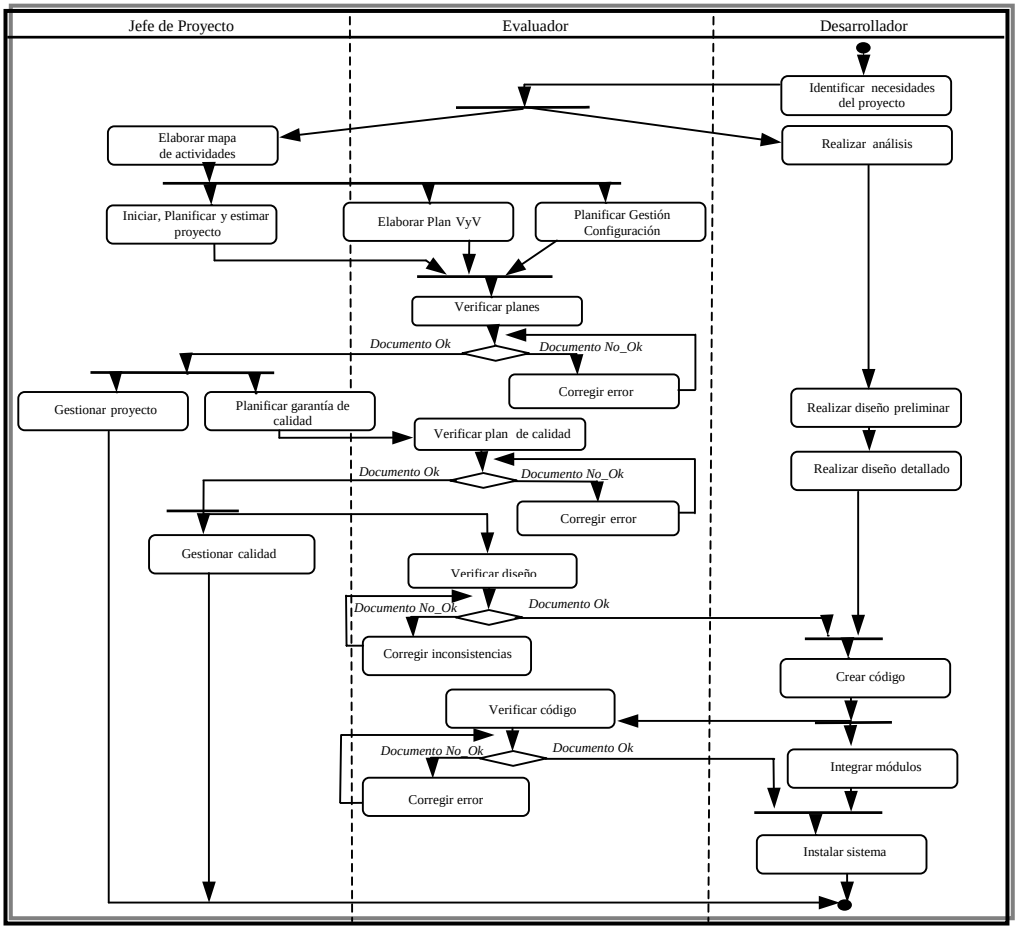


Figura 2. Diagrama de actividades del proceso software

Como se mencionó anteriormente, la red de Petri resulta de la traducción del diagrama de actividades, donde cada estado es traducido a un *lugar* en la red de Petri.

En la red generada se visualizan dos tipos de relaciones entre los distintos estados que componen la red: las relaciones de los estados involucrados en cada grupo de actividad y, la comunicación entre los distintos grupos de actividades de gestión, desarrollo y soporte (esta comunicación se representa mediante arcos dibujados con líneas gruesas, Figura 3). Por ejemplo la comunicación establecida entre el grupo de actividades de desarrollo y el grupo de actividades de gestión en las etapas iniciales del proyecto; a través de la transición T_0 que activa el paso al estado G_1, donde ocurre la primera actividad perteneciente al grupo de gestión. Este formalismo posibilita la visualización y análisis de las secuencias y concurrencias de las actividades correspondientes a los diferentes grupos y la comunicación entre los mismos a un nivel de detalle global.

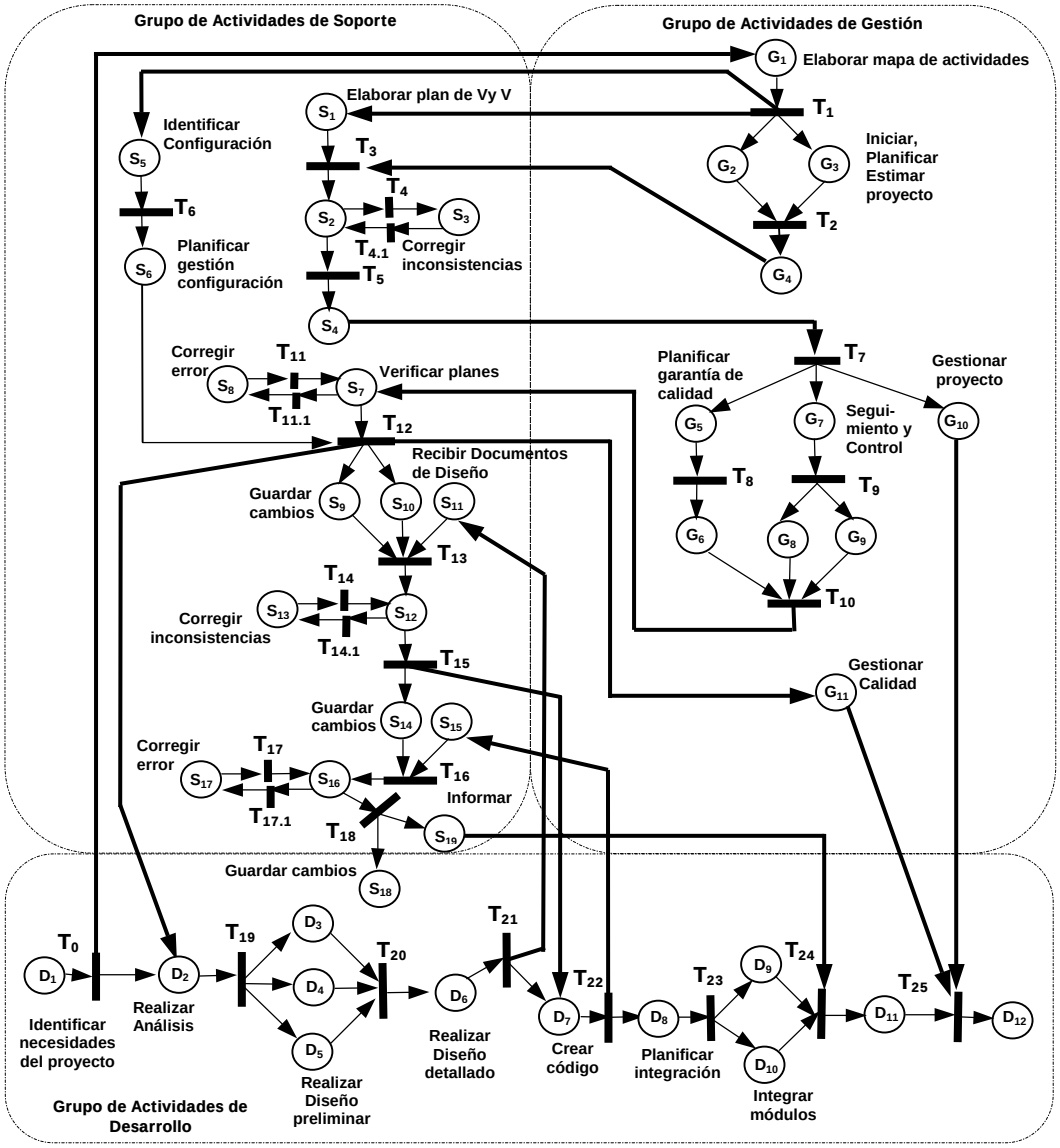
3.3. Verificación de Propiedades

Una vez traducidos todos los diagramas, se pueden usar los métodos de análisis de red de Petri [13], por ejemplo, grafos de alcanzabilidad y cubrimiento, matriz de ecuaciones, simplificación y técnicas estructurales para evaluar las propiedades del modelo: *alcanzabilidad de estados*, *vivacidad*, *acotación*, *seguridad o persistencia* [5]. En particular en AToM^3 se han implementado las basadas en el grafo de alcanzabilidad y cubrimiento.

La verificación del modelo desarrollado tiene básicamente los siguientes objetivos generales:

- Asegurar que el sistema no quede bloqueado en algún estado o actividad, es decir, que cada vez que se inicie una actividad, la misma finalice.
- Verificar aspectos de corrección y seguridad, es decir que se alcancen los estados permitidos.
- Determinar el orden de precedencia, concurrencia y secuencialidad de las actividades.

A partir de la red de Petri, se genera el grafo de alcanzabilidad que representa todas las posibles ejecuciones del sistema. Posteriormente se verifica si el sistema alcanza bloqueo, si un objeto alcanza un cierto estado, o si cierta propiedad se mantiene hasta que alguna otra propiedad se haga verdadera. La verificación se realiza aplicando model checking a las propiedades especificadas con CTL.



Transición (Ti)	Lugar (Gi, Si, Di)
T_0: Informe de necesidades terminado T_1: Enviar mapa de actividades / MCVS T_2: Iniciar plan de gestión del proyecto T_3: PGPS y plan V&V terminados T_4: Verificación NoK $T_{4.1}$:Modificaciones concluidas T_5: Verificación de planes terminado T_6: Iniciar plan de gestión configuración T_7: PGPS verificados T_8: Plan de garantía de calidad terminado T_9: Riesgos identificados T_{10}: Plan de contingencia e informe completo T_{11}: Verificación NoK $T_{11.1}$: Modificaciones concluidas T_{12}: Verificación plan concluido T_{13}: Iniciar revisión del diseño T_{14}: Verificación NoK $T_{14.1}$: Modificaciones concluidas T_{15}: Revisión del diseño concluida T_{16}: Iniciar revisión de código T_{17}: Verificación NoK $T_{17.1}$: Modificaciones concluidas T_{18}: Verificación del código concluida T_{19}: Iniciar diseño preliminar T_{20}: Iniciar diseño detallado T_{21}: Diseño terminado e iniciar código T_{22}: Código terminado T_{23}: Integración de código completada T_{24}: Iniciar instalación T_{25}: Enviar informe sobre proyecto concluido	**Grupo de Actividades de Gestión** G_1: Elaborar mapa de actividades según MCVS G_2: Definir entorno del proyecto G_3: Estimar proyecto G_4: Elaborar plan de gestión del proyecto de software (PGPS) G_5: Elaborar plan de garantía de calidad G_6: Identificar necesidades de mejora G_7: Identificar riesgos G_8: Planificar contingencias G_9: Implementar sistema de informes G_{10}: Gestionar proyecto G_{11}: Gestionar calidad del proyecto **Grupo de Actividades de Desarrollo** D_1: Identificación de necesidades del proyecto D_2: Especificar requisitos funcionales D_3: Realizar diseño preliminar D_4: Diseñar base de datos D_5: Diseñar interfaces D_6: Realizar diseño detallado D_7: Crear código D_8: Planificar integración D_9: Crear datos de prueba D_{10}: Integrar módulos D_{11}: Instalar sistema D_{12}: Guardar informes finales **Grupo de Actividades de Soporte** S_1: Elaborar plan de verificación y validación (VyV) S_2: Verificar documento PGPS S_3: Corregir inconsistencias del PGPS S_4: Guardar documento versión final S_5: Identificar configuración del sistema S_6: Planificar gestión de configuración del software S_7: Verificar plan de calidad y de contingencias S_8: Corregir error en planes S_9: Guardar documento versión final S_{10}: Preparar métricas S_{11}: Recibir Documentos de diseño S_{12}: Verificar diseño S_{13}: Corregir inconsistencias S_{14}: Guardar documento versión final S_{15}: Recibir Código S_{16}: Verificar Código S_{17}: Corregir errores S_{18}: Guardar documento versión final S_{19}: Elaborar informes

Figura 3. Red de Petri del proceso software

Algunas de las propiedades consideradas para la comprobación del modelo generado se describen a continuación:

✓La evolución del proyecto software se realiza a través del análisis de la propiedad estructural de la red modelada: *"El proceso sólo se bloquea al alcanzar el estado final"*. Por un lado se comprobaría que no hay ningún estado que satisfaga la fórmula *Deadlock AND NOT* D_{12} (ya que D_{12} es el estado final del proceso, y *Deadlock* es un predicado que es verdadero si el estado no tiene sucesores), y por otro que D_{12} es alcanzable y es final: *E(True U (*D_{12} *AND Deadlock))*. El operador *E* significa "*existe un camino en la ejecución*", y *U* es el operador "*until*". El estado inicial debe formar parte del conjunto de estados que verifican esta fórmula.

✓La precedencia y concurrencia del modelo se expresan mediante: "*Sólo una vez verificado el PGPS (*S_2*) se procede a gestionar el proyecto* (G_{10})". Esta propiedad se verifica comprobando que no hay ningún estado que satisface la fórmula *E(NOT* S_2 *U* G_{10}*)*. Esto quiere decir que no hay ninguna ejecución en la que G_{10} sea verdadero y S_2 no haya sido antes verdadero.

✓"*La actividad gestionar proyecto (*G_{10}*) termina solamente luego de realizada la instalación del sistema (*D_{11}*)*". Esto se puede verificar comprobando que el estado inicial satisface la fórmula *A (True U (*G_{10} *AND D11 AND AX (NOT* D_{11} *AND NOT* G_{10}*))*. Donde *A* es el operador "*para todos los caminos*", y *AX* significa, "*para todos los caminos, en el estado siguiente*".

En la Figura 4 se presenta como ejemplo, la comprobación de la propiedad *E(NOT* S_2 *U* G_{10}*)*. En la ventana del fondo aparece la red de Petri resultado de la transformación. En la ventana 2, parte del grafo de alcanzabilidad, que contiene 180 estados. En la ventana 3 se muestra la fórmula a comprobar, y finalmente en la 4 el resultado de la verificación.

Se han verificado distintos aspectos de diseño del proceso de software. Algunas cuestiones relevantes detectadas son las inexactitudes en la secuencia de las actividades; por ejemplo la actividad "*gestionar proyecto*" concluía inmediatamente después de realizada la actividad de verificación del plan y no tenía continuación hasta la terminación del proyecto. Esto afectaría el desarrollo del proceso, ya que los planes contienen parámetros para el seguimiento y control de la gestión y calidad del mismo. Las inexactitudes en la secuenciación de las actividades se detectan en la matriz de análisis de la red de Petri. Para resolver este obstáculo, se vuelve al diseño y se modifica el orden de precedencia de las actividades. Se cambia el diagrama de actividades y se realiza la reconversión a red de Petri. Se destaca la importancia de gestionar el proyecto desde las etapas iniciales hasta su finalización, además de la detección temprana de errores en etapas previas a la implantación.

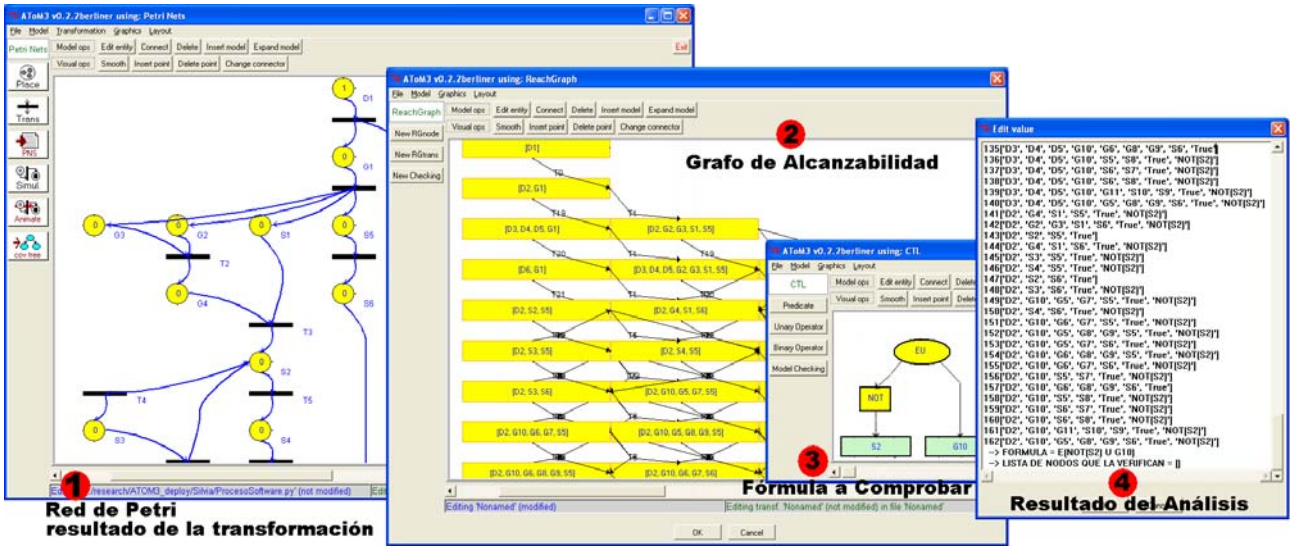


Figura 4. Comprobación automática de propiedades del proceso software

3.4. Implicaciones en el Proceso Software

En un proceso de desarrollo de software típico, suelen presentarse las siguientes situaciones: carencia de documentación de los modelos o productos obtenidos en cada una de las etapas del proceso, que dificulta la tarea de mantenimiento; que el mantenimiento se realice exclusivamente sobre el código, como producto principal obtenido al final del desarrollo; o que la verificación del producto se realice tardíamente en las etapas finales del proceso. En este trabajo se enfatiza la importancia del modelado y los modelos como los principales productos que se generan durante el desarrollo. Los modelos se generan en las etapas iniciales del desarrollo al nivel de problema o necesidad, y al nivel de solución e implantación en las etapas finales del desarrollo.

La propuesta de AToM^3 se basa en la definición de lenguajes de dominio específico, la generación de entornos de modelado para ellos, y la verificación de modelos mediante su transformación a dominios semánticos. En nuestro caso, se ha usado un subconjunto de UML para el modelado de los procesos, y una transformación a Redes de Petri, para su posterior

análisis mediante model-checking. La posibilidad de automatizar la transformación desde el lenguaje fuente, hace que los usuarios puedan utilizar los lenguajes de modelado que conocen, en vez de utilizar directamente la notación de análisis (Redes de Petri en nuestro caso).

En el modelo de proceso representado se consideran además de los aspectos propios del desarrollo de un proyecto de software, los aspectos de gestión del proyecto, gestión de configuración y control y garantía de calidad. Las actividades de gestión son responsabilidad del *gestor*, las actividades de gestión de la configuración son realizadas por el *gestor de cambios* y el control y garantía de calidad está a cargo del rol *evaluador*.

Tanto para los requisitos de usuario como para el diseño, se especifican ciertas propiedades o condiciones que el sistema debería verificar. Las condiciones deben traducirse en alguna notación para su comprobación; en esta herramienta se cuenta con CTL. Una vez que las propiedades se obtienen, pueden ser verificadas en algunos de los modelos obtenidos. Con el objeto de realizar la verificación los modelos tienen que ser traducidos, pero esta actividad podría ser transparente para el diseñador.

La herramienta utilizada en este trabajo aplica métodos formales para la comprobación. Aunque estos métodos ofrecen beneficios significativos en términos de mejora de la calidad, no son usados ampliamente debido a su alto costo y a la necesidad de contar con profesionales especialistas en algún método formal. La posibilidad brindada de especificar el sistema en un lenguaje de modelado ampliamente conocido, como es UML y luego traducir automáticamente esos modelos a un dominio semántico para realizar análisis, libera a los diseñadores de la necesidad de aprender nuevos lenguajes de especificación o metodologías para el uso de un método formal. Se propone integrar en los proyectos del software el uso de herramientas de apoyo para hacer más efectiva la aplicación de las técnicas formales.

Por último desde un punto de vista práctico los beneficios de la aplicación del enfoque y en particular el uso de AToM^3 se resume en: a) capacidad de representación gráfica y alto nivel de abstracción, b) potencialidad para realizar transformaciones de modelos en forma automática y c) posibilidad de establecer y visualizar diferentes estados por los que va atravesando el proceso software, permitiendo así el seguimiento, la detección de errores y el rediseño durante el desarrollo. Finalmente, estas ventajas permiten la mejora del proceso y su correspondencia efectiva con el proceso realizado por la organización desarrolladora de software.

4. Trabajos Relacionados

Existen diferentes tipos de modelados del proceso. Los procesos se pueden modelar a diferentes niveles de abstracción (por ejemplo, modelos genéricos versus modelos personalizados, modelos descriptivos versus modelos prescriptivos versus modelos proscriptivos). La diferenciación entre estos tipos de modelos ha sido descrita en [11]. La estructuración de los tipos de información en un modelo de proceso puede realizarse a partir de diferentes perspectivas. Se presenta la siguiente lista de perspectivas de la información comunes encontradas en la literatura:

- ✓ *Funcional*: representa cuáles elementos del proceso se están realizando, y cuáles flujos de entidades de información son relevantes a esos elementos del proceso.
- ✓ *Comportamiento*: representa cuándo (por ejemplo, secuencialidad) y bajo qué condiciones los elementos del proceso se realizan.
- ✓ *Organizacional*: representa dónde y por quién en la organización, los elementos del proceso se realizan.
- ✓ *Informacional*: representa las entidades informacionales producidas o manipuladas por un proceso, incluyendo su estructura y relaciones.

Los modelos de proceso se construyen según los lenguajes o formalismos creados para representar la información específica sobre estas características. El lenguaje más usado en la práctica es el lenguaje natural (estructurado) debido a su flexibilidad. La Tabla 1 presenta una lista de abstracciones de representación (excluyendo el lenguaje natural) organizadas según los puntos de vista mencionados previamente. Ninguna de estas abstracciones cubre todas las clases de información. Por esta razón, la mayoría de los modelos encontrados en la literatura presentan lenguajes basados en más de una de estas abstracciones. Estos modelos consideran la necesidad de la integración de múltiples paradigmas de representación (es decir, diferentes modelos de proceso), aunque esto generalmente determina una mayor complejidad en la definición del modelo [12]. Sin embargo, con AToM^3 es posible esta integración soportada por la misma herramienta.

Por una parte, como se puede observar en la Tabla 1, hay una gran variedad de notaciones que han sido usadas para modelar los procesos. Por otra parte, una variedad de enfoques multiparadigmas [4] han sido propuestos para modelar los procesos software centrados alrededor de una paradigma principal. Hay dos características principales de estos enfoques, en primer lugar, con excepción principalmente de los enfoques basados en redes de Petri temporales, todos ellos usan representaciones textuales y no temporales. En segundo lugar, la mayoría de ellos ya está en un nivel de abstracción de lenguajes de programación. Así, dificultan la modelación directa de una situación problemática, mientras que fuerzan al diseñador a tomar en cuenta muchos aspectos técnicos del formalismo de modelación [12]. Estas dos carencias han sido cubiertas en la formalización dinámica del proceso software aquí presentada. Además, de lograr cubrir las cuatro perspectivas consignadas en la modelación del proceso software a través de los formalismos orientados a objetos y temporales utilizados en este artículo (filas sombreadas en Tabla 1).

Tabla 1. Bases de los lenguajes aplicables en las perspectivas de información del proceso

Base del Lenguaje	Perspectivas de la Información
Lenguaje de programación procedimental	Funcional – Comportamiento Informacional
Análisis y diseño de sistemas, incluyendo diagrama de flujo de datos y técnica de análisis y diseño estructurado (SADT)	Funcional – Organizacional - Informacional
Lenguajes y enfoques de inteligencia artificial, incluyendo reglas y pre-/ post-condiciones	Funcional - Comportamiento
Eventos y disparadores // Flujo de control	Comportamiento
Transición de estados, redes de Petri temporales // Statecharts	**Funcional – Comportamiento -Organizacional**
Lenguajes funcionales // Lenguajes formales	Funcional
Modelación de datos, incluyendo diagramas de entidad-relación, declaraciones de relaciones y de datos estructurados	Informacional
Modelación de objetos, incluyendo tipos de clases e instancias, jerarquía y herencia	Organizacional - Informacional
Modelación cuantitativa, incluyendo técnicas cuantitativas de la Investigación Operativa y la Dinámica de sistemas	Comportamiento
Redes de precedencia, incluyendo la modelación de la Dependencia del Actor	Comportamiento - Organizacional
Lenguajes de lógica temporal, CTL, TCTL	**Funcional - Comportamiento**

5. Conclusiones

En este trabajo se ha utilizado una herramienta que permite generar, traducir, analizar y comprobar el comportamiento dinámico temporal del modelo propuesto. Hemos analizado la evolución de un modelo de procesos considerando nociones temporales, determinando cuáles son los posibles estados por los que pasa, e identificando puntos en los que se pueda tomar decisiones para evitar que el modelo alcance algún estado crítico. En particular, en este trabajo, se establece claramente el orden de precedencia y simultaneidad de los grupos de actividades considerados. Se logra precisar el orden y secuencialidad de las actividades involucradas en el proceso propuesto.

En este ejemplo donde se aplica AToM^3 al proceso de software, se percibe la facilidad de modelar separadamente cada una de las partes del proceso, pero al mismo tiempo como componentes integrados de una especificación completa. Este aporte es significativo ya que la especificación modular disminuye la complejidad de la actividad de modelación e incrementa la posibilidad de cambio y evolución de los modelos de proceso software. Además de la importancia de contar con la posibilidad de aplicar múltiples formalismos para obtener múltiples visiones de un mismo proceso, en forma gráfica, simple y efectiva.

Por último, se destaca la importancia de disponer de técnicas de modelado, transformación y de verificación formal integradas, como herramientas fundamentales de apoyo a los métodos convencionales de análisis y diseño de sistemas, especialmente para aquellos sistemas cuyo fracaso significa grandes costos.

Como trabajo futuro, pretendemos ocultar totalmente al usuario el uso de model-checking, dando los resultados de la verificación directamente en los diagramas de actividad. Al ser AToM^3 una herramienta para manejo de lenguajes visuales, la fórmula CTL se ha de especificar visualmente, aunque en este caso es más conveniente una interfaz textual. Estamos trabajando en la posibilidad de añadir una sintaxis concreta textual a los distintos metamodelos.

Referencias

[1] S. T. Acuña, A. de Antonio, X. Ferré, M. López y L. Maté, *The Software Process: Modelling, Evaluation and Improvement*. Handbook of Software Engineering and Knowledge Engineering. Vol. I (World Scientific, 2001) 193-237.

[2] AToM^3 Home page: http://atom3.cs.mcgill.ca

[3] E. M., Clarke, O. Grumberg y D. A. Peled, *Model Checking*. (MIT Press, 1999).

[4] A. Finkelstein, J. Kramer y B. Nuseibeh, *Software Process Modelling and Technology.* (Research Studies Press, 1994).

[5] E. Guerra y J. de Lara. *A Framework for the Verification of UML Models. Examples Using Petri Nets. Actas de VIII Jornadas de Ingeniería del Software y Base de Datos*. 325-334. (2003).

[6] IEEE Standard for Developing Software Life Cycle Processes, IEEE Standard 1074-1997.

[7] J. de Lara y H. Vangheluwe. *AToM^3: A Tool for Multi-Formalism Modelling and Meta-Modelling. Proc. ETAPS/FASE'02*. LNCS 2306, Springer-Verlag. 174-188. (2002).

[8] J. de Lara y H. Vangheluwe. *Computer Aided Multi-Paradigm Modelling to process Petri-Nets and Statecharts*. *Proc. ICGT'2002*. LNCS 2505. 239-253. (2002).

[9] J. de Lara y G. Taentzer. *Automated Model Transformation and its Validation with AToM^3 and AGG*. *Diagrams'2004*. Lecture Notes in Artificial Intelligence 2980, Springer-Verlag. 82-198. (2004).

[10] M. M. Lehman, *Software Engineering, the Software Process and their Support*. *Software Engineering Journal* **6**, 5 (1991) 243-258.

[11] I. R. McChesney, *Toward a Classification Scheme for Software Process Modelling Approaches.* Inform. and Soft. Technology **37**, 7 (1995) 363-374.

[12] S.-Y. Min y D.-H. Bae, *MAM nets: A Petri-net Based Approach to Software Process Modeling, Analysis and Managemen*". *Proc. of the 9th International Conference on Software Engineering and Knowledge Engineering* (1997) 78-86.

[13] T. Murata, *Petri Nets: Properties, Analysis and Applications*. *Proc. of the IEEE* **77**, 4 (1989) 541-579.

[14] Software Process Engineering Metamodel (SPEM), version 1.1, OMG Document formal/05-01-06. (2005).

[15] S. A. White (Ed.), BPML 1.0, BPMI,http://www.BPMI.org. (2003).

O Papel do CMMI na Configuração de um Meta-Processo de Produção de Software com Características Fabris: Um Estudo de Caso

José Augusto Fabri[123], André Luiz Presende Trindade[12], Márcio Silveira[4], Marcelo S. de Paula Pessoa[2]

[1]FATEC Ourinhos – [2] Universidade de São Paulo. [3]Fundação Educacional do Município de Assis – Assis SP. / [4]PUC - Pontífica Universidade Católica do Rio de Janeiro e Electronic Data System

fabri@femanet.com.br, altrindd@yahoo.com.br, marcio.silveira@eds.com, mpssoa@terra.com.br

Abstract

This work presents the relation among the CMMI model and the software meta-process production with characteristics of factory. To validate the meta-process and the relation (focus of the work) was developed a case study in the multinational Electronic Data System (EDS) – Rio de Janeiro. This company has certification CMMI5 – Software Engineering Institute (SEI – SCAMPI-A). In the case is possible to notice the main characteristics of the EDS-Rio, detaching the software production process and the process machine.

Resumo.

Este trabalho apresenta a relação entre o modelo de qualidade CMMI e o meta-processo de produção de software com características fabris. Para validar o meta-processo e a relação (foco desse trabalho) foi desenvolvido um estudo de caso na multinacional Electronic Data System (EDS) unidade Rio de Janeiro, empresa essa que possui avaliação oficial do "Software Engeenering Institute" (SEI – SCAMPI-A) CMMI-5. No estudo de caso é possível notar as principais características da EDS Rio, destacando o processo de produção de software e máquina de processo.

1. Introdução

Atualmente, o mercado de desenvolvimento de software brasileiro trava uma batalha constante na busca pela competitividade internacional, o que só pode ser alcançado com altos níveis de qualidade e produtividade. Isto pode ser comprovado ao analisar os vários programas de incentivo promovidos pelo Ministério de Ciência e Tecnologia (MCT), através da Secretária de Política em Informática (SEPIN). Um destes programas é o SOFTEX (Sociedade para Promoção da Excelência do Software Brasileiro), cujos objetivos envolvem: situar o Brasil entre os 5 maiores produtores e exportadores de software do mundo e alcançar padrão internacional de qualidade e produtividade (http://www.softex.com.br).

Além desses programas, o MCT e a SEPIN desenvolvem, periodicamente, uma pesquisa para verificar atributos de qualidade e produtividade do mercado brasileiro de desenvolvimento de software. A última delas foi publicada em 2002 e revela que, no Brasil, existem cerca de 11.000 empresas com atividades relacionadas ao desenvolvimento e comercialização de software, empregando mais de 158.000 pessoas. Cerca de 25% destas empresas possuem um programa de qualidade definido, outros 26% delas sentem a necessidade de estabelecer um programa de qualidade. Isto indica que o mercado brasileiro está tomando consciência da necessidade da qualidade em seus processos de software e, conseqüentemente, em seus produtos (MCT-SEPIN (2002) [9]).

Costa (2003) [4] apresenta uma pesquisa envolvendo as 31 empresas mais significativas, atuantes no mercado brasileiro de software com a denominação de Fábrica de Software. Seu trabalho aponta que apenas 41% delas aplicam um ciclo completo de desenvolvimento de software para seus produtos (dos requisitos à manutenção); 45% aplicam metodologia própria; 16% utilizam ferramentas de controle de projetos; 14% possuem certificação CMMI ou ISO; 13% utilizam ferramentas CASES e 10% aplicam métricas de qualidade. Assim, é válido afirmar que, apesar da tomada de consciência do mercado brasileiro em relação à produção de software, existe um grande trabalho a ser desenvolvido pela comunidade (empresas, universidades e governo) na busca dos objetivos estabelecidos pelo SOFTEX.

Paralelamente a tais fatos, existe uma discussão, nos âmbitos empresarial e acadêmico, sobre o tema "fábrica de software": afinal, o que é o processo fabril de software? Muitas empresas, erroneamente, classificam seu processo de desenvolvimento de software convencional como sendo fabril, pois a maioria delas não possui um processo que prima por produtividade e qualidade, e um processo de desenvolvimento que não atenda a estas características, não pode ser considerado fabril.

Com base nesse contexto esse trabalho apresenta a proposta de um meta-processo de produção com características fabris, relacionado-o com o modelo de qualidade CMMI, com o objetivo de verificar o papel de tal modelo dentro do meta-processo.

2. Fábrica de Software

Na literatura, é possível encontrar vários autores que trabalham, diretamente, com o conceito de Fábrica de Software, dos quais destacam-se:

Segundo Cusumano (1989) [5], o termo fábrica de software foi utilizado, pela primeira vez, na década de 1960, no Japão. Várias empresas associam o termo ao mero desenvolvimento de software; entretanto, empresas que não atendam características como: produção em larga escala; padronização de tarefas; padronização de controle; divisão do trabalho; mecanização e automatização, não podem ser consideradas fábricas de software. Para o autor, o desenvolvimento de uma fábrica implica nas boas práticas da engenharia de software aplicadas, sistematicamente.

Bemer (1969) [1], por sua vez, define Fábrica de Software como um ambiente no qual se constrói programas e se efetuam testes (ciclo de produção). Nesse ambiente devem existir ferramentas apropriadas para realizar as ações de construir e testar, medidas de produtividade e qualidade, registros financeiros mantidos por custo da programação e gerenciamento que dê subsídios à previsão e à estimativa de dados de futuros projetos.

Li et. al. (2001) [8] corroboram, afirmando que uma Fábrica de Software deve possuir um conjunto de ferramentas padronizadas para a construção de software, bases históricas para serem utilizadas no gerenciamento de projetos e, principalmente, um alto grau de reuso de código no processo de produção de software.

Baseado nas definições apresentadas, este trabalho propõe uma visão integrada do conceito de fábrica de software, visualizando-a como uma organização estruturada, voltada para a produção do produto software, totalmente, alicerçada na engenharia e com forte caracterização pela organização do trabalho, pela capacidade de modularização e reuso intensivo de componentes e pela escalabilidade produtiva.

3. Formalização dos Conceitos de Meta-Processo com Características Fabris

Para formalizar o meta-processo de produção de software com características fabris, é necessário definir, primeiramente, o conceito de processo.

Segundo Sommerville (2003) [11], um processo é um conjunto de atividades, métodos, práticas e transformações que, ordenados mesmo que, parcialmente, atingem um determinado objetivo. O processo é, assim, caracterizado como uma receita a ser seguida em projetos, os quais caracterizam sua instanciação, levando à produção de um determinado produto, no caso software.

Uma vez definido o processo, pode-se pensar em meta-processo. Reis (2002), define-o como uma estrutura que proporciona o desenvolvimento de processos executáveis para a produção de software. Os autores desse trabalho encaram meta-processo como uma receita para construir um processo.

Feiler e Humphrey (1993) [6] propõem um conjunto de atividades que configuram a estrutura básica de um meta-processo:

- Analise de requisitos do processo: deve retratar uma descrição superficial do processo a ser desenvolvido, aspectos organizacionais (relativos ao ciclo de produção) e tecnológicos de um determinado ambiente configuram os requisitos do processo;
- Modelagem do processo (projetando o processo): resulta em um modelo de processo abstrato que deve ser materializado por meio de linguagem apropriada *(process modeling language (PML)* [1]*);*
- Instanciação: resulta na produção de um processo instanciado a partir de um processo abstrato, ou seja, nessa atividade os ativos de processo (ou *templates*) são configurados, os recursos para execução do processo são alocados e o cronograma de implantação é definido. A meta-atividade de instanciação também é responsável pela configuração da máquina de processo[2].
- Simulação: permite prever problemas e estimar a duração do processo quando esse estiver em produção. Quando problemas são detectados nessa atividade, o projetista do processo deve retornar às fases anteriores para que novos ajustes sejam feitos. Nessa atividade, o

[1] Segundo, Conradi et. al. (1999) [11], as PMLs são classificadas como linguagens utilizadas para a definição e automação do processo de produção e devem dar subsídios para que o projetista descreva todas as atividades do processo.

[2] Schaefer et. al. (1999) [10] afirmam que a máquina de processo, pode ser classificada como um software com o objetivo de auxiliar na comunicação e coordenação das atividades realizadas pelos envolvidos no processo.

conhecimento do processo deve ser distribuído na organização.

- Execução: A máquina de processo (gerada pelo meta-processo) entra em operação e provê a interação entre os envolvidos e o processo de produção de software.

Um seqüenciamento das atividades, apresentando suas entradas e saídas é proposto na Figura 1. Nessa Figura é possível verificar que, após efetuar a descrição superficial do processo fabril (ciclo de produção, ambiente tecnológico, domínio do conhecimento e modelo de processo), o arcabouço para o desenvolvimento do processo é instanciado e recheado com os conceitos de qualidade e engenharia de software para que o mesmo possa ser modelado (atividade de modelagem). Na modelagem, são definidos os diagramas de atividades do processo fabril. A atividade de instanciação define os *templates* (documentos, padrões e ativos de processo) e a máquina de processo é configurada. A simulação realiza os testes no processo e os problemas (se existirem) devem ser solucionados. Por fim, o processo de produção de software entra em execução.

Na Figura 1 existe, também, a presença da linha de persistência, tal linha evolui, paralelamente, em relação ao meta-processo e suas funções resumem a garantia da institucionalização do processo de produção e a promoção da retro-alimentação com o objetivo de melhorar a qualidade do mesmo.

É importante ressaltar que o arcabouço para a configuração do processo fabril está modelado em UML (nesse caso a UML foi utilizada como uma PML (Jäger et. al. (1999) [7]). A escolha da UML justifica-se no fato de que a mesma é difundia junto a um grande número de pessoas da comunidade científica e comercial da área de engenharia de software, o que possibilita entendimento mais facilitado e intuitivo para muitos. A visão completa do arcabouço, para configuração do processo fabril, está materializada na seção 4.

Um elemento importante na definição do meta-processo a ser considerado é a máquina de processo (Schaefer et. al. (1999) [10]). Essa máquina deve atuar, diretamente, no processo de produção de software e possui as seguintes funções: apoiar e monitorar o desenvolvimento cooperativo dos envolvidos no processo; registrar dados históricos de execução; garantir a composição e o seqüenciamento das atividades definidas no processo; gerenciar o versionamento das informações geradas pelo processo; coletar, automaticamente, os índices designados pelas métricas, gerados com a execução do processo; possibilitar a alteração do processo durante a sua execução, e; gerenciar a alocação de recursos inerentes ao processo.

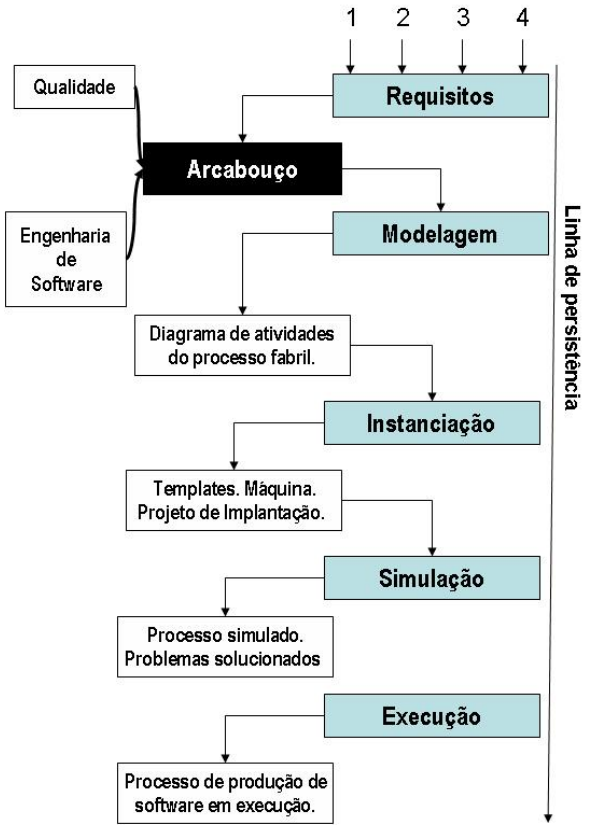


Figura 1 - Seqüenciamento das atividades do meta-processo (1 – Ciclo de Produção, 2 – Modelo de Qualidade, 3 – Modelo de processo, 4 – Domínio do conhecimento

4. A Relação entre o Meta-Processo e o CMMI

A forte relação entre o meta-processo de produção de software e o CMMI[3] encontra-se no arcabouço para configuração do processo fabril (vide Figura 1). Tal arcabouço é instanciado a partir das entradas efetuadas na atividade de requisitos: ciclo de produção, ambiente tecnológico, domínio do conhecimento e modelo de processo.

O ciclo de produção de uma fábrica de software pode ser caracterizado como:

- Fábrica de Software de Ciclo Completo: devem realizar levantamento de requisitos, modelagem de negócio, equalização do modelo de negócio[4], projeto

[3] Salienta-se que este trabalho não apresenta uma descrição formal do CMMI. Tal descrição pode ser encontrada no modelo CMMI-SE/SW (2002) [2].

[4] A equalização do modelo de negócios tem, como objetivo, verificar as especificações do modelo de negócio quanto à completude, corretitude e consistência em relação aos requisitos do cliente, bem como assegurar seu entendimento, para que as mesmas possam ser colocadas em linha de produção (atingindo a

de software, produção do código, teste modular ou unitário (os componentes são testados de forma isolada), teste de integração (os componentes são testados de forma integrada), teste de aceitação (o software é testado na presença do usuário ou cliente), gerenciamento de projetos e, por fim, a entrega do software (envolve questões como garantia do produto e treinamento dos usuários). As fábricas com este escopo devem possuir uma forte padronização dos artefatos (ou ativos de processo) pertencentes às atividades de modelagem de negócio e projeto de software. É importante ressaltar que tais atividades dependem, primordialmente, da mão de obra criativa do analista de sistemas e do arquiteto de software.

- Fábrica de Software de Ciclo Incompleto: Neste escopo, a modelagem de negócio não é caracterizada no processo fabril, ficando sob responsabilidade da fábrica apenas as atividades de equalização do modelo de negócio, projeto de software, produção de código, testes (modular, integração e aceitação), gerenciamento de projetos;
- Fábrica de Software de Ciclo Curto: Neste tipo, são contemplados às atividades de equalização dos artefatos do projeto de software (verificar se os artefatos advindos da atividade de projeto são padronizados e compreensíveis), produção de código, teste (de componente e de integrado) e gerenciamento de projetos. A modelagem de negócio, o projeto de software, o teste de aceitação e entrega é de responsabilidade do cliente. Assim, a fábrica de software passa a ser uma sub-contratada para a realização das atividades de produção e testes.

O ambiente tecnológico configura a tecnologia com a qual a fábrica de software irá operar, por exemplo: um caso de configuração como ciclo curto, produzindo código para celulares, o ambiente tecnológico da fábrica poderia ser configurado com JAVA. Ressalta-se que o ambiente tecnológico está fortemente ligado ao domínio de conhecimento (conceito, intimamente, relacionado à teoria de *Software Line Product* (SLP)[5]).

Uma outra questão importante a ser respondida na atividade de levantamento de requisitos do meta-processo é: Com qual modelo de processo[6] a fábrica de software irá operar? Esta definição influenciará o comportamento dinâmico do processo fabril, uma vez que o mesmo define como as atividades do processo estarão se relacionando sob a ótica temporal.

Efetuadas as entradas propostas na Figura 1 é possível acessar e instanciar o arcabouço para produção do processo fabril (recheado com as Áreas de Processo (PAs) do CMMI – representando o conceito de qualidade do meta-processo) e, posteriormente, modelá-lo. Tal arcabouço é apresentado na Figura 2. Ressalta-se que, nesse trabalho, a modelagem do arcabouço foi realizada considerando a representação por estágio do CMMI, escolha esta devida à empresa objeto do estudo de caso ser assim configurada.

Supondo que uma determinada organização opte por configurar uma fábrica de software de ciclo completo, ela irá instanciar a classe *fábrica de software de ciclo completo,* tal classe herda os atributos e os métodos (atividades do processo) da classe *fábrica de software*, que, por sua vez, está associada às classes *persistência*, *tecnologia*, *domínio do conhecimento* e *modelo de processo*. A classe *fábrica de software de ciclo completo* possui uma relação de dependência com a classe *fábrica de software de ciclo incompleto*, e esta, por sua vez, também possui a mesma relação com a classe *fábrica de software de ciclo curto*. Esse fato determina que as classes de *ciclo incompleto* e *curto* sejam instanciadas no momento em que a de *ciclo completo* é instanciada. Sendo, assim, também é possível concluir que a classe *fábrica de software de ciclo curto* pode ser instanciada de forma independente das classes *de ciclo incompleto* e *completo.*

Já a *classe fábrica de software* possui os métodos (atividades do processo) *produzir código*, *testar software*, *gerenciar projeto*, *gerenciar configuração* (PA do nível 2), *garantir qualidade do processo e do produto* (PA do nível 3) e *focar o processo da organização* (PA do nível 3).

A classe persistência (advinda da linha de persistência da Figura 1), é composta dos métodos institucionalizar o processo gerenciado e institucionalizar o processo definido. Esta classe se relaciona com as metas genéricas e específicas do modelo CMMI/SW. As práticas genéricas de tal têm como objetivo prover a institucionalização (nesse trabalho denominado persistência) das metas específicas definidas em cada PA de cada nível de maturidade.

atividade de projeto de software), mitigando os riscos de parada na produção e quebra na produtividade. A equalização da modelagem de negócio é uma atividade optativa e deve ser realizada quando a modelagem do negócio e projeto de software são realizados por pessoas diferentes.

[5] Uma linha de produto de software é um conjunto de sistemas que usam software intensivamente, compartilhando um conjunto de características comuns e gerenciadas, que satisfazem as necessidades de um segmento particular do mercado, e que são desenvolvidos a partir de um conjunto comum de ativos principais e de uma forma pré-estabelecida" ((Clements e Northrop (2002)) [3].

[6] Segundo Sommerville (2003) [11], "um modelo de processo de software é uma representação abstrata do processo de software". Os modelos são representações genéricas e algumas vezes chamados de paradigmas de processo.

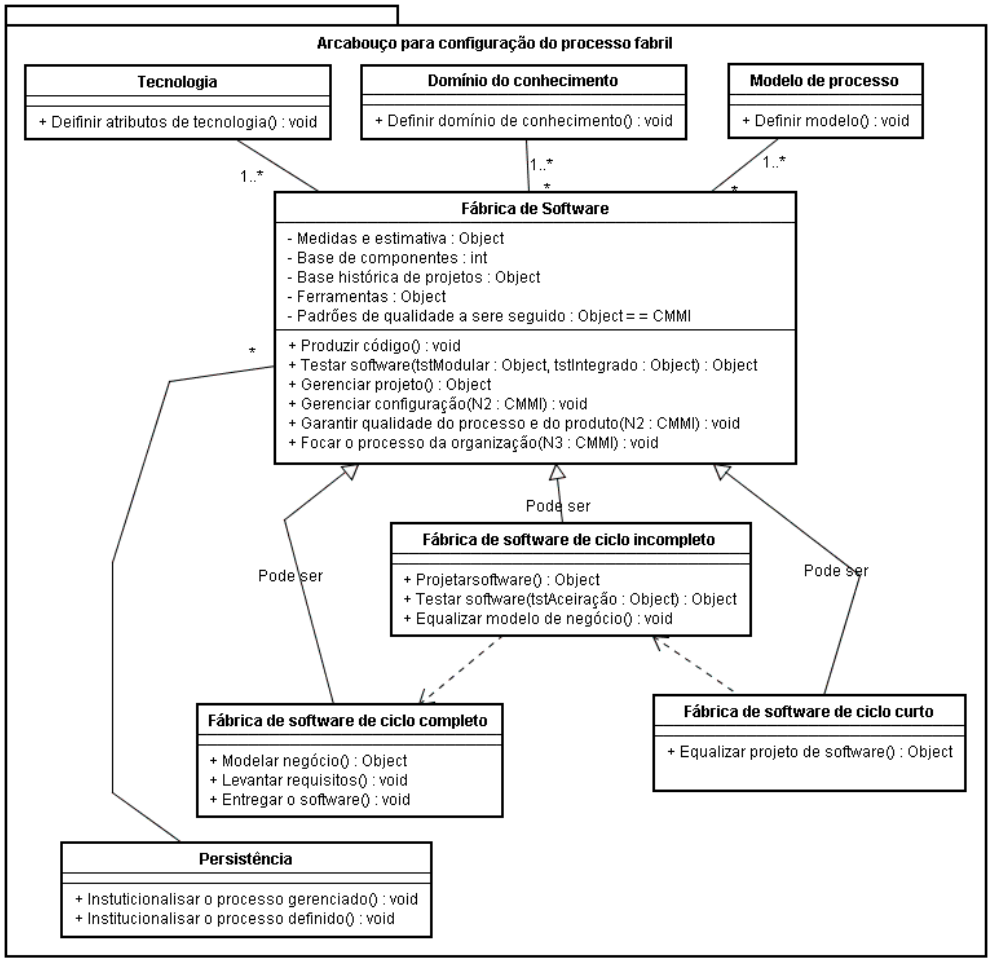


Figura 2 – Diagrama de Classe[7] representado Arcabouço para configuração do processo fabril

Uma visão completa da modelagem do arcabouço para configuração do processo fabril pode ser verificada na Figura 3. Nessa Figura nota-se a presença das classes, métodos e atributos transcritos nos parágrafos anteriores; dos diagramas de atividades que devem ser modelados pela atividade de modelagem do meta-processo (vide Figura 1).

É importante salientar que as classes que configuram o arcabouço foram "recheadas" com as PAs dos níveis de maturidade do CMMI (de 2 à 5). Algumas PAs foram classificadas como atividades do processo (representada pelos métodos) e outras foram classificadas como sub-atividades (representadas pelo diagrama de atividades).

Como pode ser visto nessa seção, a relação entre meta-processo e CMMI ocorre, fortemente, nas atividades de requisitos (quando o arcabouço é definido) e modelagem, já a atividade de instanciação aprofunda a construção do processo por meio da configuração dos *templates* e da máquina de processo.

Uma questão deve estar passando pela mente do leitor desse trabalho: As empresas classificadas no nível de maturidade 1 do CMMI podem instanciar o meta-processo e

[7] Ressalta-se que os atributos que possuem relevância no arcabouço estão ilustrados na classe fabrica de software, as demais classes estão congregando apenas os serviços.

galgar o nível 2? Sim, essas empresas devem instanciar a classe que reflita o seu ciclo de produção (fábrica de software de ciclo completo, ciclo incompleto ou ciclo curto); definir o domínio de conhecimento, ambiente tecnológico e modelo de processo e, por fim, modelar, instanciar, simular e executar as atividades e sub-atividades configuradas como N2 na Figura 3.

É interessante que as atividades configuradas como N3, N4 e N5 fiquem condicionadas à configuração prévia das atividades N2 (regra da representação por estágio).

5. Validação da Relação entre Meta-Processo e CMMI: Um Estudo de Caso

O estudo de caso apresentado nessa seção segue as considerações metodológicas apresentadas por Yin (2005) [12]. Tal estudo foi efetuado na Electronic Data Systems Corporation (EDS), unidade do Rio de Janeiro – Brasil. A EDS Rio possui certificação oficial CMMI, nível de maturidade 5, obtida em abril de 2005.

A EDS é uma multinacional com uma base empresarial instalada em 60 paises e coleciona cerca de 35.000 clientes ao longo de seus 42 anos de existência, empregando algo em torno de 120.000 funcionários, sendo que 33.000 são engenheiros de softwares. Em território brasileiro, a EDS emprega 550 Engenheiros de Software no estado do Rio de Janeiro, 1000 no estado de São Paulo e em torno de 100 em Santa Catarina.

A EDS provê serviço de TI em 4 grandes áreas: consultoria de negócio; fornecimento de infra-estruturas (servidores, suporte e manutenção de redes); aplicações (desenvolvimento e manutenção de sistemas de informação, incluindo a questão dos sistemas legados) e gerenciamento de processos de negócios que não sejam classificados como o "*core business*" de seus clientes (exemplo: gerenciamento de um "*call center*" em uma empresa da industria automobilística).

O estudo de caso focou, basicamente, a área de aplicações da EDS. Tal área possui unidades operacionais instaladas nos Estados Unidos, América Latina, Canadá, Europa, África, Oriente Médio, Ásia (Oceano Pacífico) e Austrália. Essas unidades compartilham do mesmo modelo funcional denominado pela empresa como "*Global Delivery Model*" (GDM). Por meio desse modelo é possível compartilhar toda a metodologia para o desenvolvimento de projetos (salienta-se que os projetos também podem ser desenvolvidos de forma compartilhada e colaborativa).

Dentro do GDM é possível encontrar a estrutura organizacional para o gerenciamento estratégico de capacidades, denominado, pela EDS, como "*Strategy Capability Management*" (SCM), cujos objetivos são definir as disciplinas ou "*capabilities*" que os centros de desenvolvimento (ou unidades produção), espalhados nos 5 continentes, devem possuir para que os produtos sejam desenvolvidos e entregues com um alto padrão de padronização e qualidade.

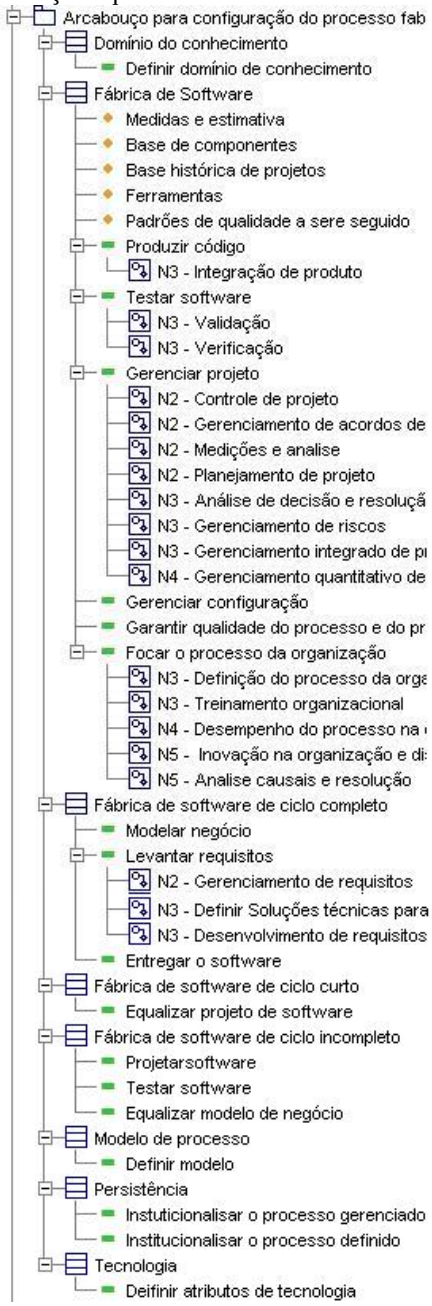


Figura 3 – Visão completa da modelagem UML do meta-processo

Sob a ótica desse trabalho, a estrutura organizacional para o gerenciamento de estratégia da EDS, o SCM, é responsável pela execução das atividades do meta-processo propostas na seção 3.

No estudo de caso foi possível constatar que a SCM definiu que a EDS Rio opera como uma fábrica de software de ciclo curto, para determinados projetos (nesse trabalho

denominado como conjunto C de projetos) e uma fábrica de software de ciclo incompleto para outros projetos (nesse trabalho como conjunto I de projetos).

O perfil tecnológico e o domínio de conhecimento na qual a unidade irá operar também é definido pela SCM.

A configuração do processo é de responsabilidade do grupo de processo de engenharia de software (*Software Engineering Process Group* (SEPG)). O SEPG pode ser configurado como local ou global. Em nível local, o SEPG é composto por funcionários ligados à própria unidade de produção. Já em nível global, o SEPG reúne os líderes dos SEPGs de algumas unidades.

Analisando o contexto apresentado nos últimos parágrafos é possível constatar que a SCM, juntamente, com SEPG definiram os requisitos do processo (tanto para projetos caracterizados como C quanto para projetos caracterizados como I). Ao definir os requisitos é possível constatar que o processo de produção de software da EDS Rio possui duas visões: ciclo curto (Figura 4) e ciclo incompleto.

A visão de ciclo curto foi instanciada a partir da definição do ciclo de produção para os projetos do tipo C e a visão de ciclo incompleto foi instanciada a partir do ciclo de produção para os projetos do tipo I. A instanciação de tais visões foi intermediada pelo arcabouço para configuração do processo apresentado na Figura 2. É importante ressaltar que o modelo de processo utilizado em ambas as visões é caracterizado como incremental.

O gerenciamento global é feito pela unidade de produção contratante (geralmente, locada nos Estados Unidos) e tem, como objetivo, controlar o projeto executado pela unidade de produção contratada (unidade que atende um cliente interno, o que ocorre na maior parte dos projetos estabelecidos na EDS Rio).

Na Figura 4 é possível constatar a existência de duas formas de gerenciamento de projetos: local e global. O gerenciamento local ocorre de forma paralela às atividades instanciadas a partir do arcabouço para a configuração do processo, e tem, como objetivo, planejar e controlar a execução do projeto de software dentro do ambiente da unidade de produção (em nosso caso, no Rio).

Apresentados os requisitos para a configuração do processo de software da EDS e elaborada a visão alto nível do processo de software, visão esta estabelecida com a ajuda do arcabouço para configuração do processo fabril, faz-se necessário executar a atividade de modelagem do meta-processo (atividade essa apresentada na Figura 1). Salienta-se que, para fins ilustrativos, será apresentada apenas a modelagem da atividade de teste, sub-atividade de verificação (respeitando as diretivas estabelecidas no modelo CMMI).

A Figura 5 apresenta a sub-atividade de verificação dividida em metas específicas (representada pelas siglas SG 1, SG 2 e SG 3) e práticas específicas, estas caracterizadas como tarefas dentro da sub-atividade (siglas SP x,y onde x caracteriza a meta específica e y caracteriza a prática). A Tabela 1 apresenta a descrição da meta específica 1 e da prática específica 1.1. A descrição das demais metas e práticas podem ser encontradas em CMMI-SE/SW (2002) [2].

Após executar a atividade de modelagem do meta-processo, faz-se necessário instanciar o processo modelado com o objetivo de configurar os *templates* e a máquina de processo. Para fins ilustrativos, será apresentada, apenas, a configuração do *template* que apresenta a lista dos produtos selecionados para verificação (vide Figura 6).

Após modelar os *templates* e os ativos de processo a máquina de processo deve ser configurada.

A máquina de processo da EDS, denominada *Global Solution Management System* (GSMS), integra todas as atividades do processo, o fluxo de trabalho, os artefatos para desenvolvimento de cada atividade ou tarefa e, por fim, provê mecanismo para alimentar a base histórica de projetos (base esta configurada de forma local e global). No GSMS é possível desenvolver toda a atividade de gerenciamento de projeto e acessar o repositório de boas práticas, onde estão armazenados os possíveis artefatos de projeto que podem ser reutilizados.

Algumas características arquitônicas da máquina de processo podem ser verificadas no Quadro 1.

É importante salientar que os recursos para execução e o cronograma para implantação do processo não serão apresentados nesse trabalho, pois o processo já se encontra institucionalizado na EDS. Pelo mesmo motivo, a simulação também não será mostrada.

Por fim, é possível constatar que o meta-processo, adere ao desenvolvimento e a institucionalização do processo da empresa objeto do estudo de caso. A EDS, como um todo, está configurada nos três ciclos de produção (*curto, incompleto* e *completo*), os ciclos *curto* é visto, nitidamente, na Figura 4. O ciclo completo também pode ser instanciado pela EDS-Rio, o que ocorre com menor freqüência. Geralmente, tal ciclo é instanciado por unidades de produção localizadas nos EUA, promovendo assim o conceito *offshore*.

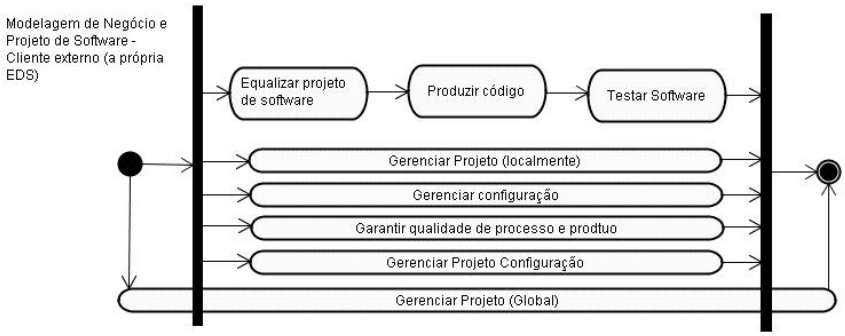


Figura 4 - Processo de Produção de Software EDS - Visão do Ciclo Curto

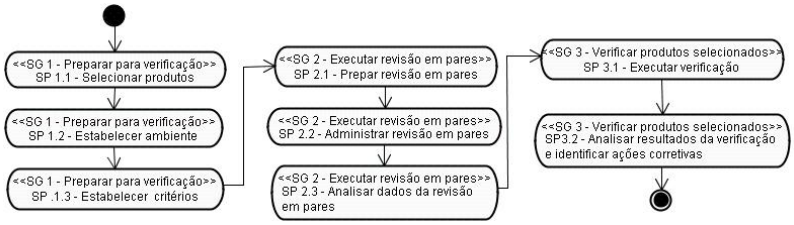


Figura 5 - Modelagem da sub-atividade de verificação (respeitando as diretivas do CMMI)

SG 1	Tem como objetivo assegurar que as providências para a verificação estejam embutidas nos produtos, requisitos, planos de desenvolvimentos e programas. A verificação incluí a seleção, inspeção, teste e demonstração dos produtos.		
	Descrição	Produtos de trabalhos	Sub-práticas
SP 1.1	Esta prática seleciona os produtos que serão verificados e os métodos usados na verificação de cada produtos.	Lista de todos os produtos selecionados para verificação. Método de verificação para cada produto selecionado. Entradas: Produtos que serão verificados Saídas: Lista dos produtos com os métodos de verificação. (a especificação da entrada e saída não estão contempladas no CMMI)	Identificar produtos para verificação. Identificar os requisitos para satisfazer o produto selecionado. Identificar os métodos de verificação que estão disponíveis para o uso. Definir os métodos de verificação para ser usado por cada produto selecionado. Submeter para integração com o plano de projeto a identificação do produto para ser verificado, os requisitos para ser satisfeitos e os métodos para ser usados.
Responsável: <<campo utilizado para definir o papel do responsável>>			

Tabela 1 – Descrição das metas e práticas específicas que compõem a modelagem de sub-atividade de verificação [2].

6. Conclusão

Esse trabalho apresentou o papel do CMMI na configuração de um meta-processo de produção de software com características fabris. Com as considerações efetuadas no trabalho foi possível concluir que o meta-processo absorve um modelo de qualidade, nesse caso o CMMI, quando o arcabouço é instanciado. Além de absorver um modelo de qualidade, os conceitos de ciclo de produção, ambiente tecnológico, domínio de conhecimento e a forte presença da engenharia de software estão imersos no meta-processo. O papel do arcabouço para configuração do processo fabril, nesse contexto, é congregar, de forma generalista, tais conceitos (vide Figura 7).

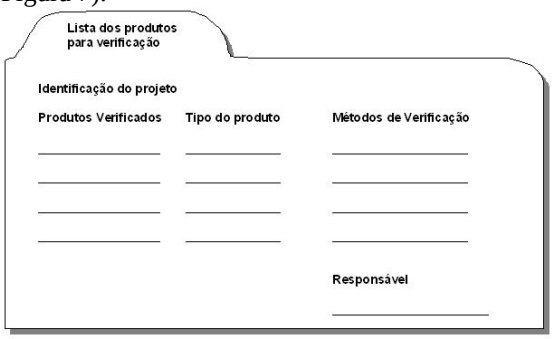


Figura 6 - Exemplo de um template

Além de todas as considerações feitas nesse trabalho, existe uma questão que não foi respondida: O meta-processo pode ser executado para a configuração de um processo de produção de software para micro-empresas (por exemplo: uma empresa que agrega três pessoas?). Sim, basta que o ciclo de produção da empresa seja definido (*completo*, *incompleto* ou *curto*). Após definir o ciclo de produção as classes *fábrica de software*, *tecnologia*, *domínio do conhecimento* e *modelo de processo* devem ser instanciadas. Ao instanciar essas classes, a tecnologia, o domínio do conhecimento e o modelo de processo são definidos. Por fim, a necessidade de modelar apenas as atividades das classes instanciadas deve ser contemplada.

Quadro 1 – Aspectos arquitetônicos da máquina de processo

Opção para configuração e seqüenciamento das atividades de processo, respeitando as instanciações das classes *fábrica de software, fábrica de software de ciclo completo, incompleto e curto*.
Opção para que o gerente do projeto possa estabelecer uma matriz de responsabilidades.
Opção para que os envolvidos com o processo possam informar o tempo de realização de uma determinada tarefa, para um determinado projeto de software.
Por fim, a máquina de processo fornece a visão administrativa da Fábrica de Software, através de informações sobre o andamento do processo, tais como: históricos de execuções e as principais métricas de qualidade (medida em quantidade de desvios de execução de processo, erros em testes e satisfação do cliente) e produtividade.

Um outro aspecto interessante deve ser ressaltado nesse trabalho: É na atividade de simulação que ocorre a distribuição do conhecimento relacionado ao processo de produção de software, é nesse momento que o processo e a máquina de processo estão configurados e o treinamento, dos envolvidos, no processo é realizado. É de extrema importância que o treinamento seja feito com a máquina de processo configurada, pois ela é quem congrega tudo o que foi feito nas atividades de requisitos, modelagem e simulação do meta-processo, provendo subsídios para que as atividades de processo sejam seguidas e a base de conhecimento de projeto alimentada.

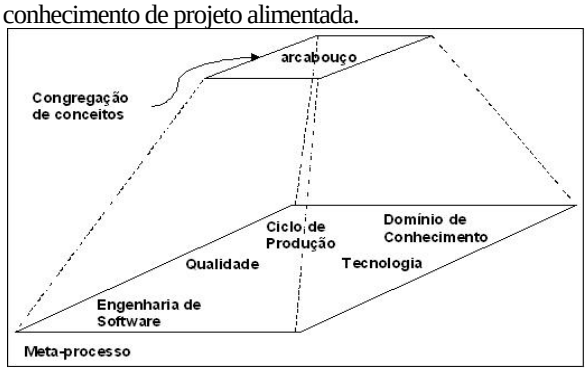


Figura 7 – Papel do arcabouço no meta-processo

Enfim, os autores propõem que novos estudos de casos sejam desenvolvidos, em trabalhos futuros, para verificar a total aderência do meta-processo fabril a outros modelos de qualidade.

Referências

[1] Bemer, Robert. W.; Position Papers For Panel Discussion: the economics of program production. In: "Information Processing" – no. 68, vol. II; Holanda: North-Holland Publ.Co., 1969.

[2] CMMI-SE/SW (2002). Capability Maturity Model Integration for System Engineering and Software Engineering CMMI-SE/SW, V1.1. CMU/SEI-2002-TR-001 ESC-TR-2002-002. disponível em http://www.sei.cmu.edu/cmmi/.

[3] Clements, Paul; Northrop, Linda. (2002) "Software Product Line: Practices and Partterns". Boston. Addison Wesley.

[4] Costa, Ivanir. (2003) "Contribuição para o aumento da qualidade e produtividade de uma fábrica de software através da padronização do processo de recebimento de serviços de construção de softwares" - 174 pag.; Tese (Doutorado) Apresentada ao Departamento de Engenharia de Produção da Universidade de São Paulo; São Paulo: PoliUSP.

[5] Cusumano, Michael A. (1989) Software Factory: A Historical Interpretation; In: "IEEE Software" - vol. 6, No 2, pp. 23-30. March/April.

[6] Feiler, P.; Humphrey, W. (1993) Software Process Development and Enactment: Concepts and Definitions. In: "International Conference on the Software Process", ICSP Proceedings**.** Berlin, Germany: IEEE Computer Society Press.

[7] Jäger, D., Schleicher, A., and Westfechtel, B. (1999) Using UML for software process modeling. In "Proceedings of the 7th European Software Engineering Conference Held Jointly with the 7th ACM SIGSOFT International Symposium on Foundations of Software Engineering". Toulouse, France, September 06 - 10.

[8] Li, C.; Li, H.; Li, M. (2001); A Software Factory Model Based on ISO 9000 e CMM for Chinese Small Organization. In: "Second Asia-Pacific Conference on Quality Software (APAQS'01)". Hong Kong: IEEE, December.

[9] MCT-SEPIN (2002) - Secretária de Política de Informática do Ministério da Ciência e Tecnologia; "Relatório de Qualidade e Produtividade no Setor de Software Brasileiro" Número 4 - ISSN 1518-112X; Brasília. Disponível em www.mct.gov.br/sepin

[10] Schaefer, Wilhelm; Fuggetta, Alfonse; Gobart Claude, Jahnke Jens. (1999) Architectural Views and Alternatives. "Lecture Notes in Computer Science. Software Process: Principles, Methdology, and Techonology". Derniame, Jean-Claude; Kaba, Badara Ali; Wastell David Eds. Vol. 1500.. ISBN. 3-540-65516-6. Pages 95-116. Springer-Verlag.

[11] Sommerville, Ian. (2003) **"**Engenharia de Software" - 6ª Edição; Addison Wesley, 2003.

[12] Yin, Robert K. (2005) "Estudo de Caso: Planejamento e Método". 3 ed. traduzida. Bookman. Porto Alegre.

[11[CONRADI, R.; JACCHERI, M. L. (1999); Process Modeling Language. In: DERNIAME, Jean-Claude; KABA, Badara A.; WASTELL, David. "Software Process: Principles, Methodology, and Technology". Series: Lecture Notes in Computer Science, Vol. 1500.

Utilización de un método ad hoc de evaluación para el mejoramiento de procesos con MoProSoft

Verónica Martinez[1],Yessica Gómez[1] , Hanna Oktaba[2], Angélica Urrutia[1] ,Rodolfo Villarroel[1]

Universidad Católica del Maule
Departamento de Computación e Informática
Avenida San Miguel 3605, Talca, Chile
1{vmatinez, ygomez, aurrutia, rvillarr }@spock.ucm.cl
[2]{ho}@fciencias.unam.mx

Resumen

Muchas pequeñas y medianas empresas en Chile desean el mejoramiento de sus procesos de software, pero sólo pocas han tomado el rumbo de una evaluación por medio de los modelos y estándares existentes debido a lo costos asociados a la evaluación e implantación de procesos de mejora. Actualmente, estas pequeñas organizaciones de software buscan mejorar sus procesos de desarrollo para mejorar la calidad de los productos, mejorar la productividad del equipo desarrollador y reducir los tiempos de desarrollo; sin embargo, no son muchas las que conocen la manera de lograr estas mejoras. En este artículo se define un metodo ad hoc para su utilización en una pequeña empresa dedicada al desarrollo y mantenimiento de software en Chile. Se presentan los resultados de la evaluación realizada, basándose en el modelo MoProSoft y en el método EvalProSoft. Con esta experiencia se pudo constatar el estado actual de las prácticas de los procesos de MoProSoft utilizando un método ad hoc y que podría ser generalizada a todas las pequeñas y medianas empresas desarrolladoras de software.

1. Introducción

Desde hace muchos años se ponía de manifiesto el rudimentario estado de desarrollo de software en comparación con otras disciplinas, y su difícil crecimiento en un entorno de demanda creciente de sistemas de mayor tamaño y más sofisticado.

Un objetivo claro es la mejora de la calidad. Es ampliamente aceptado que la calidad de un sistema de software se rige por la calidad de los procesos realizados en su desarrollo. Han sido varias las instituciones y consorcios que han tenido como motivación la mejora de los procesos de desarrollo y mantenimiento de software. Entre los resultados de estas entidades se encuentran recapitulaciones y catálogos de buenas prácticas y modelos de procesos basados en buenas prácticas, dando lugar a una rama de investigación como es la evaluación y mejora del proceso de software.

Existiendo la necesidad de fortalecer a la industria del software de Iberoamérica, principalmente en el fortalecimiento del nivel de competitividad de las pequeñas y medianas empresas desarrolladoras de software, es importante alcanzar niveles internacionales en capacidad de procesos mediante la utilización de un modelo de procesos y de evaluación apropiado para la industria del software de iberoamérica.

En Chile es fundamental darle prioridad a la micro, pequeña y mediana empresa para un mayor desarrollo del país, pues ofrecen más del 70% del empleo del país [1]. Por lo tanto, se les debe dar la importancia que merecen para mejorar su productividad y la participación en el mercado nacional e internacional.

Muchas pequeñas y medianas empresas en Chile desean el mejoramiento de sus procesos de software, pero sólo pocas han tomado el rumbo de una evaluación por medio de los modelos y estándares existentes debido a lo costos asociados a la evaluación e implantación de procesos de mejora. Por lo tanto, surge como una buena respuesta la aplicación de un modelo no costoso en su adopción y que sirva como base para alcanzar evaluaciones exitosas con otros modelos o normas.

Las universidades, a la vez, tienen el deseo de participar en este tipo de mejoramiento, pero la relación universidad-empresa es algo que se debe cultivar y las estadísticas muestran que hay una gran separación que aún cuesta ir superando. Según el ENTI 2006 [4], realizado entre marzo y julio de 2006 en Chile, solamente el 5% de las empresas consultadas menciona a las universidades como fuente de innovación, en contraste con el 15% de las empresas

europeas. Debido a todos los aspectos mencionados, en este artículo se presenta una experiencia relacionada con la aplicación del modelo MoProSoft, dentro de una método ad hoc, en una pequeña empresa desarrolladora de software. Se utilizó MoProSoft debido a que este modelo está pensado para ser utilizado en este tipo de empresas. Este modelo es fácil de entender, fácil de aplicar, no es costoso en su adopción, y va a ser considerado como un punto de referencia para identificar los elementos que faltan por cubrir.

Las siguientes secciones de este artículo contienen: Una mirada global a los modelos de procesos existentes y la justificación del uso de MoProSoft en el método que se propone (Sección 2), los pasos del método ad hoc utilizado en una pequeña empresa desarrolladora de software (Sección 3), los resultados de la aplicación del método (Sección 4), las conclusiones y trabajo futuro (Sección 5).

2. Modelos de procesos

Se entiende como proceso de software a un conjunto de procedimientos, métodos, equipos y herramientas que están a disposición de las personas con el propósito de concebir, desarrollar, instalar y mantener un producto de software. La evaluación de procesos es un examen disciplinado de los procesos utilizados por una organización, de acuerdo con algunos criterios, para determinar la capacidad que tienen estos procesos de ser ejecutados dentro de metas de calidad, de costo y de tiempo. El propósito es caracterizar la práctica actual, identificando fortalezas, debilidades y la habilidad de los procesos para controlar o evitar las causas de baja calidad, desviaciones en costos o planificación. Para que ésto sea consistente y se repita, debe definirse un modelo de referencia.

A nivel internacional existen diferentes modelos, estándares y sistemas de certificación para las empresas desarrolladoras de software, los cuales tienen su propio método de evaluación o certificación, por lo tanto, sus pasos y resultados finales pueden tener ciertas variaciones.

En el caso de Latinoamérica, muchos países han comenzado a preocuparse de la calidad de los procesos asociados a desarrollo y mantenimiento de software para sus empresas, de manera de mejorar sus procesos y que le permitan generar productos de calidad con el fin de ser competitivas a nivel nacional e internacional. De esta manera, se logra una diferenciación competitiva, donde el mercado reconoce la evidencia de la calidad de los productos y servicios proporcionados. Sin embargo, los países latinoamericanos reconocen que los modelos de mejoramiento que proveen las organizaciones como el SEI (*Software Engineering Institute*) e ISO fueron diseñados para su aplicación en grandes empresas, por lo tanto, para empresas pequeñas puede ser muy difícil sobrellevar un proyecto de mejoramiento que signifique invertir grandes cantidades de dinero, tiempo y recursos. Es por este motivo que se han generado esfuerzos importantes que permitan aumentar la competitividad de las pequeñas y medianas empresas, por medio de propuestas que reconocen basarse en los modelos y estándares anteriores, con el fin de mejorar los procesos críticos de la producción de software en las pequeñas y medianas empresas, como los procesos de operación y los procesos de entrega de los productos y servicios.

Las propuestas más sólidas son: MoProSoft (México), MPS BR (Brasil) y Avansoft (Colombia). Las dos primeras propuestas están orientadas a pequeñas y medianas empresas, mientras que Avansoft está muy orientado a proyectos grandes y tiene una formalidad extrema que lo convierte en inflexible.

El modelo MPS BR está basado en ISO/IEC 12207:2002, CMMI e ISO/IEC 15504:2003, cuyo objetivo es que las PyMES obtengan un nivel de madurez 2 o 3 a un costo que sea accesible para ellas. Esta solución plantea dos modelos:

- MN MPS: Modelo de Negocio para la mejora del proceso de software. Permite definir los elementos e interacciones involucrados para lograr una certificación de la empresa por medio de la implementación del modelo MR MPS.
- MR MPS: Modelo de Referencia para la mejora del proceso de software. Esta implementación puede ser personalizada para una empresa o conjunta para un grupo de empresas, de manera de lograr costos menores.

Teniendo presente una implementación futura gradual de CMMI en las pequeñas empresas brasileñas, el modelo establece 7 niveles con áreas de procesos basadas en este modelo.

MoProsoft, en cambio, considera tres categorías de procesos que reflejan la estructura de una organización: Alta Dirección, Gerencia y Operación. Cada categoría está asociada con uno o más procesos. Las categorías con sus procesos y subprocesos son los siguientes:

- Alta Dirección
  - Gestión de Negocio
- Gerencia
  - Gestión de Procesos
  - Gestión de Proyectos
  - Gestión de Recursos
    - Recursos Humanos y Ambiente de Trabajo
    - Bienes Servicios e Infraestructura

        - Conocimiento de la Organización
- Operación
    - Administración de Proyectos Específicos
    - Desarrollo y Mantenimiento de Software

En cada proceso se definen los roles responsables por la ejecución de las prácticas, los cuales se asignan al personal de la organización de acuerdo a las habilidades para un buen desempeño y a las capacitaciones realizadas.

Se ha decidido utilizar MoProsoft en nuestro método ad hoc debido a que ha sido adoptado por el gobierno mexicano y está lo suficientemente probado de manera exitosa para la realidad mexicana. Ejemplos exitosos en su implantación se pueden mencionar a las empresas Sistemas de Gestión Administrativas S.C., Magnabyte, E-Genium,S.C. y Arquitectura en Tecnología de Mexico, S.A. de C.V. De ahí nuestro interés para utilizarlo en Chile en una empresa desarrolladora de software, interesada en mejorar sus procesos, pero no con los suficientes recursos para someterse a una evaluación del tipo CMM o ISO.

3. Método ad hoc utilizado

A continuación, se presenta el método ad hoc que se aplicó a una pequeña empresa desarrolladora de software. La Figura 1 muestra el mapa conceptual con los pasos de nuestro método.

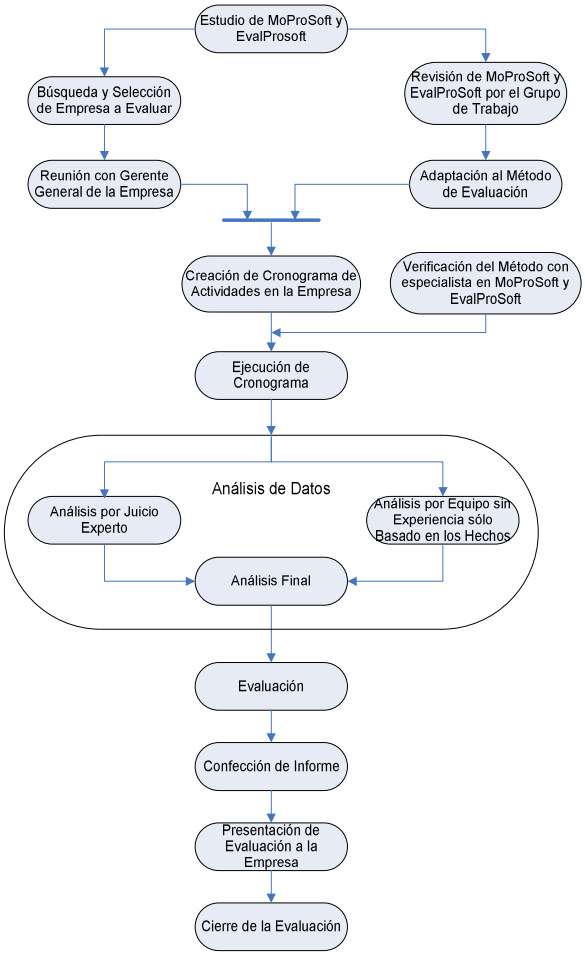


Figura 1. Método ad hoc utilizado

3.1. Estudio de Moprosoft y Evalprosoft

Es el primer paso necesario para poder comenzar todo el proceso de evaluación. En el cual se realiza el

estudio del modelo de procesos para la industria del software, MoProSoft y el método de evaluación de procesos para la industria del software, EvalProSoft. Este conocimiento es vital para poder realizar una evaluación de procesos basada en MoProSoft.

3.2. Búsqueda y Selección de la Empresa a Evaluar

Se buscan posibles empresas que se dediquen al desarrollo y mantenimiento de software. Luego, se plantea el proceso de evaluación para, finalmente, seleccionar a la empresa que acepta el proceso.

3.3. Revisión de Moprosoft y Evalprosoft por el Grupo de Trabajo

En esta actividad se revisa tanto el modelo de procesos como el método de evaluación, pero además se explica en que consiste todo este proceso, lo que se busca y quiere lograr, como se menciona a continuación:

- Revisión y compresión por parte de los participantes del grupo de trabajo del modelo de proceso Moprosoft y método de evaluación Evalprosoft.
- Aclarar conceptos clave y necesarios para entender el modelo de procesos MoProSoft y los objetivos a cumplir en la empresa.
- Definir roles dentro del grupo para la aplicación del método.

3.4. Reunión con Gerente General de la Empresa

Se Realiza una reunión con el Gerente General de la empresa que permita:

- Mostrar los objetivos que se persiguen lograr con todo el proceso de evaluación.
- Señalar lo que se debe y no se debe esperar de la actividad de evaluación sus beneficios.
- Hacer hincapié en que lo que se busca evaluar es el proceso, y no a las personas, para evitar cualquier sesgo en las respuestas. Esta actividad es clave para asegurar la adecuada utilización del test de evaluación.

3.5. Adaptación al Método de Evaluación

Luego de realizar todo el estudio del modelo de procesos MoProSoft y el método de evaluación EvalProSoft, se revisa punto por punto cada objetivo que persigue el modelo con el fin de realizar preguntas enfocadas a cada actividad que permita cumplir con el objetivo en las áreas de proceso del modelo.

Se crea un test de evaluación dividido por áreas de procesos, Alta Dirección, Gestión y Operación. Adaptar el test considerando la simplificación de las nomenclaturas utilizadas por Evalprosoft para el facil entendimiento en la empresa.

Se realiza adecuación del método de evaluación de acuerdo a la realidad del grupo evaluador, dado a que no se cuenta con evaluador certificado, no pudiendo cumplir con lo explicitado en la primera etapa de Evalprosoft , y se fue adecuando además a la realidad en la que se encontraba el grupo de trabajo y a la empresa.

3.6. Creación de Cronograma de Actividades

Se construye un cronograma con todas las actividades, detallando todas las reuniones que comprenden la evaluación, los participantes de ella, fecha de realización y los objetivos que se quieren lograr en cada una de las siguientes actividades:

- Solicitud de los siguientes datos:
  - Nombre de la organización.
  - Organigrama de la organización.
  - Plan de mejora (si lo tienen).
  - Inventario de proyectos dentro del alcance de la evaluación.
  - Nombre del representante de la organización.
  - Motivación par realizar la evaluación, qué los lleva a realizarla y qué esperan de ella.
  - Acuerdo de confidencialidad general sobre el control de la información resultante de la evaluación.
  - Duración de la evaluación, fecha de inicio y final.
  - Nombre de los encargados por áreas de procesos.
- Reunión con encargado de Alta Dirección.
- Reunión con encargado de Gestión.
- Reunión con encargado de Operación.
- Visita a las instalaciones.
- Presentación y entrega de informe de evaluación.

3.7. Verificación del método con especialista en Moprosoft y Evalprosoft

Se establece una reunión de trabajo con una especialista, en este caso particular con la Dra. Hanna Oktaba, autora del Modelo de Procesos a utilizar, para el apoyo directo en la revisión y guías para la aplicación de MoProsoft en una pequeña y mediana

empresa. También se interactúa con la especialista Claudia Alquicira en la ciudad de Mexico.

3.8. Ejecución de Cronograma

El cronograma contempla los siguientes puntos:

- Solicitud de los datos de la empresa, mencionados en la sección 3.6.
- Reunión aplicada al área de Proyectos, específicamente al Jefe de Proyectos.
- Reunión aplicada al área de Operación, específicamente al encargado de Proyectos y al encargado del Desarrollo.
- Reunión aplicada al área de Desarrollo, específicamente al Ingeniero de Desarrollo especializado en Web, al Analista-Programador y al Ingeniero de Desarrollo.
- Reunión aplicada al área de Alta Dirección, realizada al Gerente Comercial y Gerente de Proyectos (que también es socio de la empresa).

En cada una de estas reuniones con el integrante del grupo de trabajo adecuado relacionado con las categorías mencionadas en MoProSoft: Alta Dirección, Gerencia y Operación, dependiendo del área que se trataría en la reunión.

Con el test construido para la evaluación, el cual está dividido de acuerdo a las tres categorías mencionadas, se controla que se cumplan los objetivos pedidos en los procesos de cada categoría.

3.9. Análisis por Juicio Experto

Se procede a realizar un análisis de las entrevistas que permitan verificar el cumplimiento de los ítems que son necesarios en cada nivel. Este análisis fue realizado por los académicos integrantes del grupo de trabajo, los cuales tienen la experiencia y experticia en el campo laboral.

3.10. Análisis por Equipo sin Experiencia sólo Basado en los Hechos

Se realiza el mismo tipo de análisis anterior, pero por parte de los integrantes del grupo de trabajo que no tenían la experiencia en lo que es el campo laboral (alumno realizando tesis de ingeniería y un alumno con beca de investigación).

3.11. Análisis final

Después de que cada grupo de trabajo (juicio experto y equipo sin experiencia) realiza sus análisis respectivos se llega a un análisis final en conjunto, discutiendo y entendiendo cada uno de los puntos de vista de cada integrante del grupo y acordando el análisis correspondiente, aceptando las diferencias y llegando a acuerdo.

3.12. Evaluación

La idea de este trabajo fue realizar una evaluación inicial del nivel de capacidad alcanzado en sus procesos de desarrollo y/o mantenimiento, el cual se deriva de la calificación que se obtenga de los ítems correspondientes en cada uno de los atributos por nivel. La calificación de estos atributos se puede ver en la Tabla 1. Esta tabla, que se define en el proceso de evaluación EvalProSoft, indica que el grado del cumplimiento del atributo del proceso se califica usando una escala ordinal.

Estos porcentajes que podemos apreciar en la Tabla 1 son los obtenidos de cada uno de los ítems que se requieren cumplir por nivel de capacidad en cada uno de los procesos. Para poder cumplir el nivel debe cumplir ampliamente el atributo del nivel que se encuentra (entre 50 y 85%) y completamente el del nivel anterior (entre 85 y 100%).

Tabla 1. Calificación de los atributos de los procesos

N	No alcanzado	0-15% del alcance
P	Parcialmente alcanzado	> 15 % hasta 50 % del alcance
A	Ampliamente alcanzado	> 50 % hasta el 85 % del alcance
C	Completamente alcanzado	> 85 hasta el 100 % del alcance

La tabla 2 muestra la calificación de los atributos correspondientes para alcanzar cada nivel.

Nivel / Atributo	1	2	3	4	5
1.1 Realización del proceso	A	C	C	C	C
2.1 Administración de la realización	-	A	C	C	C
2.2 Administración del producto de trabajo	-	A	C	C	C
3.1 Definición del proceso	-	-	A	C	C
3.2 Implantación del proceso	-	-	A	C	C
4.1 Medición del proceso	-	-	-	A	C
4.2 Control del proceso	-	-	-	A	C
5.1 Innovación del proceso	-	-	-	-	A
5.2 Optimización del proceso	-	-	-	-	A

Tabla 2: Calificación del nivel de capacidad del proceso.

En total son 6 niveles de capacidad de procesos (del 0 al 5). Cada nivel presenta atributos con ítems que lo caracterizan, los cuales corresponden a cada uno de los niveles de capacidades de los procesos de desarrollo (Proceso Incompleto, Proceso Realizado, Proceso

Administrado, Proceso Establecido, Proceso Predecible y Proceso Optimizado).

Para el cálculo del grado de cumplimiento del atributo se utiliza la formula:

*% cumplimiento = ((Nº de respuestas afirmativas)*100)/total de prácticas del atributo*

Una vez obtenido el nivel de capacidades de cada uno de los procesos, se puede obtener el nivel de madurez de la organización que corresponde al menor de los máximos niveles obtenidos en los procesos evaluados.

3.13. Confección de Informe

La confección del informe se realiza una vez terminado todo el proceso de ejecución del cronograma, el análisis de datos y la evaluación. Este informe presenta las fortalezas y debilidades de cada proceso y las oportunidades de mejora a la empresa evaluada.

3.14. Presentación de Evaluación a la Empresa

Presentación del informe con los resultados del proceso de evaluación, para su discusión, aceptación y priorización de sugerencias de mejora y establecimiento de un plan de acción.

3.15. Cierre de la Evaluación

Creación documento de conformidad de evaluación realizada.

4. Resultados de la aplicación del método ad hoc

A continuación, se presentan los resultados del nivel de capacidad en que se encuentran los procesos de producción y/o mantenimiento de software en una pequeña empresa de desarrollo de software. La empresa analizada se dedica a proveer productos de software que satisfacen las necesidades del mercado agrícola a nivel nacional e internacional.

El propósito del procedimiento de evaluación es determinar el nivel de madurez de los procesos de producción de software de la empresa, en relación al modelo MoProSoft, y orientarla hacia un camino ordenado de mejoramiento continuo.

Se utilizaron cada uno de los pasos presentados en la sección anterior. Sin embargo, falta la presentación de los resultados en la empresa evaluada. Por motivos de espacio sólo se mostrarán los resultados obtenidos de la aplicación de nuestra metodología.

En la Tabla 3 se puede ver un resumen de manera porcentual del grado de cumplimiento de cada atributo, lo que más adelante en la Tabla 4 se ve el nivel de capacidades en que se encuentran cada uno de los procesos. La simbología utilizada en las Tablas 3 y 4 para los procesos de MoProSoft es la siguiente:

- GN: Gestión de Negocio.
- GPR: Gestión de Procesos.
- GPY: Gestión de Proyectos.
- GR: Gestión de Recursos.
- RHAT: Recursos Humanos y Ambiente de Trabajo.
- BSI: Bienes, Servicios e Infraestructura.
- CO: Conocimiento de la Organización.
- APE: Administración de Proyectos Específicos.
- DM: Desarrollo y Mantenimiento de Software.

Tabla 3. Resumen de cumplimiento de las prácticas de los atributos de procesos

% de cumplimiento de prácticas nivel 1	Procesos								
	GN	GPR	GPY	GR	RHAT	BSI	CO	APE	DM
	63,33	27,27	67,74	20	33,33	46,15	17,65	70,73	65,22

Una vez obtenido el nivel de capacidades de cada uno de los procesos (Tabla 4), se puede obtener el nivel de madurez de la organización, el cual está definido como el menor nivel de capacidades obtenido por todos los procesos de la organización.

En este caso la empresa obtiene un nivel alcanzado de cero, lo que implica una oportunidad de mejora.

Tabla 4. Nivel de capacidades de los procesos de la organización

	Procesos								
	GN	GPR	GPY	GR	RHAT	BSI	CO	APE	DM
Nivel	1	0	1	0	0	0	0	1	1

Para una mayor comprensión del cumplimiento de las prácticas de los atributos de procesos mostrados en las Tablas 3 y 4, se puede considerar el proceso GPY (Gestión de Proyectos).para el nivel 1. Este proceso debe cumplir con los ítems que requiere el atributo de realización del proceso. Es importante destacar que se comienza analizando desde el nivel 1. Si cumple un determinado nivel, entonces se prosigue con el nivel siguiente; en caso contrario, no se justifica analizar los niveles superiores. Considerando la totalidad de ítems para este atributo, la empresa logra alcanzar un 67,74%. Se puede concluir que alcanza el nivel 1 de capacidad en su proceso, por lo tanto, se puede seguir evaluando lo que cumple del siguiente atributo para el nivel 2, debido a que fue capaz de alcanzar nivel 1.

Para la realización del informe se incluyen las fortalezas y debilidades de cada proceso para una mejor comprensión por parte de la empresa. La Tabla 5 muestra las fortalezas y debilidades del proceso Gestión de Procesos..

Tabla 5. Fortalezas y Debilidades en Gestión de Procesos

Fortalezas	Debilidades
• Existe un calendario en cada proceso. • Existe documentación de los procesos. • Existe un responsable de procesos. • Se asigna y notifica a los responsables de procesos.	• No existe un plan de adquisición y capacitación. • No existe un plan de evaluación. • No existe un plan de manejo de riesgos. • No existe un responsable de gestión de procesos. • No existe un plan de contingencia. • No se realiza capacitación a la organización en los procesos.

5. Conclusiones

Este artículo no resuelve todos los problemas para el mejoramiento de procesos en la empresa analizada. Sin embargo, es sólo uno de los primeros pasos en la dirección correcta, debido a que se enfoca en un problema importante y aporta una guía para su solución al plantear un método ad hoc que utiliza el modelo de procesos MoProSoft.

El procedimiento realizado con MoProSoft ha sido útil como una guía de referencia que nos permitió considerar los procesos importantes de una pequeña empresa desarrolladora de software. De esta manera, se ha podido revisar los procesos de software de la organización y establecer sus fortalezas y debilidades, de manera de establecer oportunidades de mejoramiento y un plan de acción.

El uso y adaptación de EvalProsoft permitió la evaluación de la capacidad de los procesos de una pequeña empresa y sin demasiado gasto en dicho proceso, en comparación con otros modelos cuyas evaluaciones son muy costosas.

La mejora de la calidad de los procesos de desarrollo y mantenimiento de software dentro de las pequeñas empresas chilenas es algo muy anhelado. Por lo tanto, la adopción de un modelo y métodos adecuados es de vital importancia, debido a que permitirá la apertura a nuevos cambios en los procesos del desarrollo y mantenimiento de software con la finalidad de mantener o mejorar las oportunidades de competitividad de la empresa. Así, se conseguirá el agente de cambio que se quiere para enfrentar de mejor forma una realidad crecientemente competitiva.

Además esta experiencia puede servir de base para otros grupos de trabajo que no cuentan con evaluadores certificados en Moprosoft y Evalprosoft.

Como trabajo futuro se establecerá un plan de acción que permita mejorar las capacidades de la empresa evaluada. Además, el realizar mayores evaluaciones permitirá no sólo aplicar el método ad hoc, sino que también proponer mejoras que puedan adaptarse completamente a las pequeñas y medianas empresas de nuestro país.

6. Agradecimientos

Este trabajo forma parte del proyecto COMPETISOFT (506PI0297), financiado por CYTED.

7. Referencias

[1] CORFO. Guía de Programas para Pymes. Corporación de Fomento, Gobierno de Chile, agosto de 2006.

[2] De la Villa, Manuel; Ruiz, Mercedes; Ramos, Isabel. Modelos de Evaluación y Mejora de Procesos: Análisis Comparativo. Proceedings of the V Workshop on Decision Support in Software Engineering (ADIS2004), Málaga, España.

[3] EAFIT. 2005. El Proceso de Desarrollo de Software: Una Visión desde la Academia y la Industria. Grupo de investigación en Ingeniería de Software, Universidad EAFIT, Colombia.

[4] ENTI. 2006. Estudio Nacional sobre Tecnologías de Información. Centro de Estudios de Tecnologías de Información. Pontificia Universidad Católica de Chile. http://www.ceticuc.cl/estudios/enti_2006

[5] ISO. 1998. ISO/IEC TR 15504-1:1998 Information Technology – Software Process Assesment – Part I: Concepts and Introductory Guide.

[6] Mutafelija, B.; Stromberg, H. Systematic Process Improvement using ISO 9001: 2000 and CMMI. Artech House Computing Library, 2003.

[7] Oktaba, Hanna; Alquicira, Claudia; Su Ramos, Angélica. Modelo de Procesos para la Industria del Software MoProSoft. Versión 1.3. Agosto de 2005. Secretaría de Economía, México.

[8] Pino, F. J.; García, F. O.; Ruiz, F.; Piattini, M. Adaptación de las normas ISO/IEC 12207:2002 e ISO/IEC 15504:2003 para la evaluación de la madurez de procesos software en países en desarrollo. Revista IEEE América Latina, Vol. 4, Nº 2, Abril 2006.

[9] Serrad, S. Evolution of the Frameworks Quagmire. IEEE Computer Magazine, Vol. 34, N° 7, Julio 2001, pp. 96-98.

[10] Wang, Y.; Court, I.; Ross, M.; Staples, G.; King, G.; Dorling, A. Towards Software Process Excellence: A survey report on the best practices in the software industry. ASQ Journal of Software Quality Professional, Vol. 2, Nº 1, 1999, pp. 34-93.

[11] Weber, K.; Rocha, A. Modelo de Referência para Melhoria de Processo de Software: una abordagem brasileira. Proceedings of the QUATIC 2004, 2004, pp. 73-78.

[12] West, M. Real Process Improvement Using the CMMI, Auerbach Publications, 2004.

Perfil UML 2.0 para Aplicaciones de Monitoreo Ambiental

Adriana B. Urciuolo
Universidad Nacional de la Patagonia S. J.B.
Ushuaia, Tierra del Fuego, ARGENTINA
urciuolo@tdfuego.com

Rodolfo J. Iturraspe
Universidad Nacional de la Patagonia S.J.B.
Ushuaia, Tierra del Fuego, ARGENTINA
iturraspe@tdfuego.com

Ezequiel Moyano
Universidad Nacional de la Patagonia S. J.B.
Ushuaia, Tierra del Fuego, ARGENTINA
emoyano@infovia.com.ar

Abstract

Los sistemas del dominio de aplicaciones ambientales (DAA) cobran cada día mayor importancia, pero aún resulta incipiente el desarrollo de modelos generales y herramientas para su especificación. Si bien los diagramas de UML resultan adecuados para modelar diferentes tipos de aplicaciones, no contemplan los diferentes aspectos particulares a modelar en aplicaciones ambientales. Una ventaja de UML reside en los mecanismos que incorpora para definir extensiones para dominios específicos que colectivamente permiten obtener Perfiles. UML 2.0 facilita la definición de un Perfil a través de un nuevo mecanismo de extensión: el metamodelo. En el presente trabajo se analiza el metamodelo para un subdominio del DAA: aplicaciones de monitoreo ambiental (SMA). A partir del mismo se definen estereotipos y restricciones que permiten obtener un Perfil para aplicaciones SMA. La extensión definida se describe a través del caso de estudio Monitoreo de aguas.

1. Introducción

El estudio de los sistemas de información ambiental adquiere cada vez mayor importancia, dada la función esencial que cumplen en la toma de decisión para el manejo y aprovechamiento sustentable de los recursos naturales y el medio ambiente [17]. No obstante ello, el desarrollo de modelos genéricos para el dominio de aplicaciones ambientales (DAA) utilizando medios de especificación apropiados a las particularidades del mismo, aún es incipiente.

El nuevo campo de la Informática ambiental aplica los conocimientos de las ciencias de la información y la ingeniería a la comprensión y solución de los problemas de los sistemas ambientales [18].

Uno de los requerimientos básicos de estas aplicaciones es el *manejo de la complejidad*, debida principalmente a ciertas características de la información ambiental, tales como la representación espacial de variables físicas y su registro temporal. Aplicaciones típicas del dominio son las relativas al monitoreo ambiental, las cuales definen un subdominio (SMA), donde además del problema de la representación geográfica, debe modelarse el registro de extensas series de tiempo correspondientes a la medición de diversas variables ambientales y a la observación de fenómenos de la naturaleza [7].

El modelado de software con adecuadas herramientas para la definición de los conceptos específicos del dominio es muy necesario y conveniente para el desarrollo de sistemas grandes y complejos. UML es un lenguaje de modelado de propósito general que cubre un amplio espectro de diferentes dominios de aplicación, siendo además una notación que puede ser fácilmente extensible de manera estandarizada [15]. Aunque los conceptos de modelado generales de UML son apropiados para un amplio rango de aplicaciones, en algunos dominios puede resultar muy conveniente una especialización adicional o extensión de estos conceptos, para permitir una versión más refinada de los conceptos y técnicas específicas del dominio [11].

En ese sentido, una de las ventajas de UML son los mecanismos que incorpora para especializar las formas genéricas de sus elementos de modelado a variaciones específicas de aplicaciones de un dominio. Estas variaciones, colectivamente proveen una capacidad para obtener "Perfiles" UML, los cuales empaquetan terminología y subestructuras de un dominio de aplicación particular. En algunos dominios como EDOC y el Tiempo Real [8] ya se ha identificado la necesidad de tales requerimientos, desarrollando RPFs para perfiles UML [15].

Un Perfil es un Paquete estereotipado que contiene elementos del modelo que han sido ajustados para un

propósito específico utilizando mecanismos de extensión. El Perfil puede asimismo especificar el subconjunto del metamodelo que extiende. La versión de UML 2.0 incorpora un nuevo concepto de Perfil, introduciendo la utilización del metamodelo como mecanismo de extensión [2].

Además de los perfiles estandarizados en el contexto de OMG, los perfiles pueden ser especializados para manejar subdominios y más aún, se pueden especializar perfiles para proyectos individuales dentro de una organización para tratar óptimamente las tecnologías específicas de proyectos y sus necesidades [15].

En este trabajo se presentan modelos basados en UML 2.0 para la especificación de las actividades correspondientes al monitoreo de fenómenos ambientales, problemas típicos y recurrentes del dominio DAA. Como solución a los requerimientos de las aplicaciones mencionadas, se describe un Perfil UML 2.0 para el subdominio definido por las mismas, el cual consiste de dos partes: en la primera se define el metamodelo para las aplicaciones de monitoreo ambiental, así como la semántica y elementos relacionados de cada metaclase; en segundo lugar se mapean los conceptos de dichas aplicaciones a elementos UML [1] [4]. Este mapeo contiene información sobre cada concepto SMA, el elemento base UML que representa cada concepto y el estereotipo que extiende la metaclase, de modo que pueda utilizarse la terminología específica del dominio.

En la Sección 2 se presentan las características de los perfiles de UML 2.0, su evolución y se discute sobre extensiones y perfiles de UML existentes para aplicaciones del DAA; en la sección 3 se analizan los conceptos del metamodelo para el subdominio de aplicaciones de monitoreo ambiental (SMA) para los cuales resulta conveniente definir extensiones de UML; la sección 4 presenta un perfil de UML 2.0 para estas aplicaciones y en la sección 5 se definen las conclusiones.

2. Los Perfiles de UML 2.0 y su aplicación en el DAA

2.1. Concepto de Perfil UML

Un perfil UML es un conjunto predefinido de estereotipos, valores etiquetados, restricciones e íconos de notación que colectivamente especializan UML para un dominio específico o proceso. Un Perfil no extiende UML agregando nuevos conceptos básicos. En su lugar, provee convenciones para aplicar y especializar UML estándar a un ambiente o dominio particular [6]

El mecanismo de Perfiles ha sido específicamente definido para proveer una forma de extensión "liviana" al estándar UML. En UML 1.1. se utilizaron los estereotipos y valores etiquetados como mecanismos de extensiones basadas en strings que podían adicionarse a los elementos del modelo UML de manera flexible. En subsecuentes revisiones de UML, la noción de Perfil fue definida para proveer mayor estructura y precisión a la definición anterior.

2.2. Evolución del concepto de Perfil en UML 2.0

Los Perfiles UML se definieron originalmente en la versión 1 de UML y, si bien fueron ampliamente utilizados, resultaba difícil determinar si su aplicación era correcta debido a que su definición era un tanto ambigua [2]. La nueva versión UML 2.0 mejora substancialmente su definición, especificándose de forma más clara las relaciones permitidas entre los elementos del modelo a especificar y el uso de las metaclases de un metamodelo dentro de un Perfil UML.

La forma de considerar las extensiones ha cambiado ligeramente en UML 2.0. Existe un nuevo mecanismo de extensión: el metamodelo. Un metamodelo es el mecanismo subyacente que define el lenguaje para expresar un modelo. UML 2.0 permite entonces otra forma de extender UML mediante adiciones al metamodelo existente.

El paquete Profiles de UML 2.0 define una serie de mecanismos para extender y adaptar las metaclases de un metamodelo cualquiera (y no sólo el de UML) a las necesidades concretas de una plataforma. El metamodelo es el lugar donde se puede agregar características, tales como terminología específica, cambios a los símbolos UML y nueva información semántica. [2].

En otras palabras, la definición actual de Perfil es más restrictiva, dado que provee un límite a lo que puede hacer un Perfil. No puede contener cambios a un metamodelo existente, solo puede extenderlo usando extereotipos o restricciones.

2.3. Los estereotipos en UML 2.0

Según las especificaciones de UML 2.0, los estereotipos son los mecanismos más importantes de extensiones del metamodelo [2]. Un estereotipo define cómo una metaclase existente puede ser extendida, permitiendo el uso de terminología específica del dominio [11]. Está basado en un elemento del modelo existente y puede introducir valores adicionales, una nueva representación gráfica y una lista de restricciones aplicados a su uso. Comparten los atributos, asociaciones y operaciones de su clase base, sin embargo pueden presentar otras restricciones y diferente semántica.

2.4. Perfiles para dominios específicos

Los dominios de aplicación a menudo requieren librerías de modelos predefinidos, sin el requerimiento de un lenguaje de especificación adaptado al mismo. OMG recomienda el uso de perfiles sólo en aplicaciones donde se necesita semántica específica. En tal sentido se identificaron los siguientes tipos de aplicaciones donde el uso de Perfiles UML [11] resulta ventajoso: especialización de UML para tecnologías de implementación ("language mappings"), especialización de UML para dominios técnicos generales, especialización de UML para dominios de aplicación específicos, especialización de UML para el modelado de procesos de desarrollo de software.

En el caso de especializar UML para dominios de aplicación específicos [9], existen algunos perfiles y extensiones UML definidos en el DAA para aplicaciones geográficas. En trabajos recientes [5] se define un Perfil para bases de datos geográficas consistente en el tipo de dato "geográfico", sus restricciones asociadas y una extensión de OCL definiendo nuevos tipos básicos para expresar restricciones topológicas. Otros trabajos [10] definen estereotipos para especificación de tipos y su utilización en el campo de SIG para especificar tipos geográficos (point, polyline, polygon, circle), íconos de representación para los mismos y restricciones asociadas, así como extensiones de versiones 1.x de UML los cuales analizan formas convenientes de particularizar UML al dominio. En [12] se define una jerarquía básica para tipos de datos espaciales simples y complejos, así como notaciones visuales y herramientas para la representación de los mismos. En [14] se propone un framework para el modelado conceptual de bases de datos geográficas y estereotipos gráficos para los conceptos relevantes del dominio. No obstante, puede observarse tanto en las extensiones como en los perfiles citados, que el interés se focaliza en el problema de la representación de los objetos del mundo real en un Sistema de Información Geográfica, sin analizar el problema de modelar otras situaciones recurrentes en estas aplicaciones

En el presente trabajo, al referirse a aplicaciones ambientales, se realiza la suposición de que los objetos del dominio poseen además de características geográficas, algún tipo de comportamiento "ambiental" definido por variables físicas asociadas a los mismos, el cual puede ser monitoreado. En función de esto, más allá del problema de la representación geográfica usualmente abordado, interesa modelar el comportamiento físico recurrente de las variables ambientales.

Para ello es preciso en primer lugar, disponer de la correspondiente definición del metamodelo [4] del subdominio de aplicación (SMA) para el cual se propone un Perfil.

3. Metamodelo de aplicaciones SMA

Se define a continuación el metamodelo del subdominio de aplicaciones de monitoreo ambiental utilizando lenguaje UML. Se incluyen las entidades y asociaciones propias del dominio, según taxonomías, modelos del dominio [7] y modelos conceptuales [3] ya disponibles en otros trabajos.

Se propone un estereotipo para cada uno de los elementos del metamodelo a incluir en el Perfil. Se les asignará nombres similares, a los efectos de establecer una relación entre el metamodelo y el Perfil. [4].

Se describen en OCL sólo las restricciones del metamodelo correspondientes a estereotipos definidos en el Perfil.

3.1. Conceptos básicos utilizados en las aplicaciones del subdominio SMA

Los Sistemas del DAA se relacionan con el manejo de los datos correspondientes a los distintos componentes interactuantes del ambiente: el suelo, el agua, el aire y las especies existentes [17]. Si bien abarcan una gran variedad de problemas, en el presente trabajo se propone un Perfil para el subdominio de aplicaciones de monitoreo ambiental (problema de la medición de variables asociadas a objetos ambientales), dejando el estudio de otras aplicaciones para futuros trabajos. El subdominio de SMA corresponde a aplicaciones que manejan la recolección de información correspondiente a variables características de la naturaleza (del clima, vegetación, recursos hídricos, etc.), proveniente de mediciones realizadas por distintos medios, como sensores remotos, equipamiento automático de estaciones, etc. Esta información normalmente de gran densidad y volumen, genera extensas series de tiempo dado que los datos que representan una variable física deben ser registrados en distintas ubicaciones espaciales y a intevalos de tiempo muy pequeños, a los fines de ser útiles para el procesamiento estadístico que involucra su estudio y utilización.

El análisis de SMA implica el uso de herramientas cuantitativas para examinar sistemas como ríos, lagos y el aire a los fines de desarrollar estrategias de manejo para el control de descargas de contaminantes.

Se utilizan distintos métodos y modelos para simular el transporte y la transformación de contaminantes en los cuerpos receptores. Tanto los modelos de optimización como los métodos

cuantitativos utilizados para el soporte de políticas y estrategias de manejo ambiental, requieren de gran cantidad de variables y parámetros físicos, en el aspecto espacial y temporal, a los fines de proceder a la estimación de su comportamiento por extrapolación y predicción.

Es importante enfatizar que además del tratamiento primario de la información proveniente de distintas fuentes que se realiza en una primera fase de pre-procesamiento y formateo de datos, durante una segunda fase de análisis se procede al procesamiento estadístico y se deben aplicar métodos de cálculo, modelos de simulación, etc utilizando el conocimiento existente en las instituciones a los fines de proveer verdadero soporte a la toma de decisión

Desde esta perspectiva, el problema de la ubicación espacial de las variables es sólo un aspecto del problema general. La representación clara y sin ambigüedades en las etapas del modelado de estas aplicaciones, de variables que serán manejadas en grandes volúmenes para su análisis, se convierte en un aspecto fundamental de su tratamiento.

Se excluye en el análisis del Perfil el metamodelo correspondiente a un DAS (Data Acquisition System), por cuanto se considera conveniente modelar de forma separada el sistema de adquisición de datos que puede ser automático, manual, etc. de los conceptos de monitoreo ambiental.

Se introducen los principales conceptos relativos a las aplicaciones de SMA sobre la base de modelos conceptuales existentes.

3.1.1. Representación geográfica

El problema de su representación geográfica ha sido estudiado en diversos trabajos, incluso definiendo estereotipos gráficos para objetos geográficos [16].

Los estereotipos utilizados aseguran que cada objeto ambiental tenga su representación espacial definida en algún sistema con una determinada geometría.

En la mayoría de los métodos de modelado de SIG, los estereotipos propuestos focalizan la atención en la geometría de las entidades del mundo real. [12][16][13].

En el presente trabajo se propone la utilización de los estereotipos propuestos por Kang *et al* [5] en la definición de un Perfil para bases de datos geográficas, basado en el concepto de clase geográfica, sus restricciones de tipo y un estándar de extensiones UML.

El concepto de clase geográfica se introduce en orden a identificar claramente la geometría asociada a un objeto. Más precisamente, cada instancia de clase geográfica tiene una característica geográfica. En la práctica, una clase geográfica incluye un atributo con un tipo especial, cuyo valor puede almacenar la geometría de la entidad geográfica.

En un tipo de dato geográfico (geotype), las instancias de clases geográficas son objetos que tienen un atributo específico llamado geometría. El estereotipo <<geographic>> se asocia a cada clase geográfica. Una clase que especializa una clase geográfica, también es geográfica. El Perfil propuesto por Kang asimismo define restricciones y operadores para definir combinaciones de tipos geográficos. Para mayor detalle ver [5] y [10].

Dado que los perfiles pueden ser dinámicamente combinados y aplicados al mismo tiempo sobre el mismo modelo [2], se propone la utilización de este Perfil para la representación geográfica de los distintos elementos del modelo, definiendo en un Perfil separado los conceptos estrictamente relativos al monitoreo de variables ambientales.

3.1.2. Clases del metamodelo

Región ambiental

Una región ambiental es una región geográfica, tal como puede ser cualquier unidad jurisdiccional (país, provincia, cuenca). Pero en estas aplicaciones se le agrega la particularidad de contener objetos ambientales de algún subdominio físico que pueden ser monitoreados en puntos de control determinados. Es decir que presenta una asociación con objetos ambientales. Se propone utilizar el estereotipo <<envRegion>> para denotar la clase del metamodelo RegiónAmbiental, cuyas instancias serán clases que además de tener una geometría definida en algún sistema de representación espacial, incorporan objetos ambientales. Actualmente se considera a la cuenca hidrográfica como la unidad más apropiada de gestión ambiental, aunque esto no es una restricción, ya que pueden considerarse otro tipo de regiones. Una región puede contener a su vez regiones como se muestra en la Figura 1.

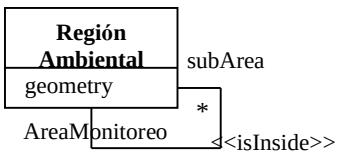


Figura 1. Metaclase RegionAmbiental

La metaclase así definida permite diferentes maneras de discretizar regiones para su monitoreo, por ejemplo dividiendo una cuenca en subcuencas o un área en celdas de una grilla.

Objetos ambientales

Son Objetos del mundo real (río, lago, suelo, vegetación, etc.) de los cuales se puede registrar el comportamiento ambiental vinculado a diferentes procesos físicos, mediante mediciones y observaciones de diferentes variables asociadas (caudal, nivel, evapotranspiración, etc.). Es decir que estos objetos, además de tener una representación geográfica, tienen variables asociadas que definen su comportamiento. En [3] se presenta una taxonomía para este tipo de objetos. Para representar clases correspondientes a este tipo de objetos, se define la metaclase ObjetoAmbiental proponiendo la utilización del estereotipo <<envObject>> en el Perfil a definir.

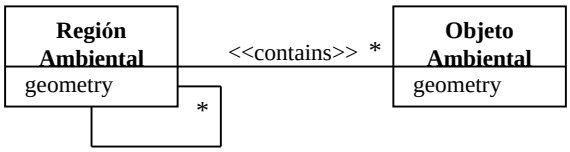


Figura 2. Metaclase ObjetoAmbiental

La asociación contains de la Figura 2 en el contexto de Sistemas que manejan información espacial, indica que la ubicación geográfica del objeto ambiental, cualquiera sea su representación, debe estar contenida dentro de la representación de la región ambiental en forma completa. Considerando que se aplicará a estas clases del modelo el Perfil geográfico [5], esta restricción puede expresarse con la operación g1>contains(g2) definida en la extensión de OCL propuesta en dicho trabajo de la siguiente forma:

Context ObjetoAmbiental inv:
self.geometry-> forAll (g|RegAmb.allInstances->exists (reg|reg.geometry->contains(g)))

Variable Ambiental Temporal

En estas aplicaciones normalmente se almacena información temporal relacionada con el comportamiento de los objetos ambientales (río, atmósfera, etc.) correspondiente a la medición de variables físicas asociadas a ellos, tales como niveles de agua registrados, caudales líquidos y sólidos, parámetros ambientales de calidad de aguas tales como ph, sodio, etc., clima tales como temperatura, humedad, precipitación, etc. Estas variables pueden ser representadas geográficamente como campo continuo o discreto [14].

Para representar clases cuyas instancias corresponden a las variables físicas asociadas a los objetos ambientales que presentan una serie de tiempo asociada correspondiente al registro histórico de sus mediciones y una unidad de medida, se define la metaclase VariableAmbiental para la cual se propone el estereotipo <<envVar>>.

Punto de control

Las variables físicas que definen el comportamiento de los objetos ambientales se registran en algún punto geográfico donde los datos son adquiridos de distintas formas, por ejemplo mediante sensores de estaciones automáticas o por medios manuales. El modelo conceptual para variables físicas medidas [3] propone desacoplar la sección de control del objeto ambiental como un objeto geográfico separado. Por ello se define la metaclase PuntoControl y se propone el estereotipo <<controlPoint>>.

En una aplicación de monitoreo ambiental, la existencia de puntos de control para la medición de las variables asociadas a los objetos ambientales constituye una restricción. Debe existir al menos uno para posibilitar una medición. No obstante figurar la restricción de multiplicidad en el metamodelo, se define en OCL por constituir una restricción para el Perfil.

Context ObjetoAmbiental inv:
self.puntoControl -> size() >= 1

Para mayor flexibilidad, el modelo permite el monitoreo de variables en un punto geográfico sin estación fija o con estación instalada. Se adiciona a una metaclase Estación el comportamiento específico del instrumental instalado, con la restricción de mantener la misma ubicación que PuntoControl, utilizando el estereotipo <<estacion>>. Se define la restricción en OCL:

Context Estación inv:
self.puntoControl.ubicación = self.ubicación

La Figura 3 muestra la metaclase PuntoControl y la metaclase Estación definiendo comportamiento adicional a la anterior.

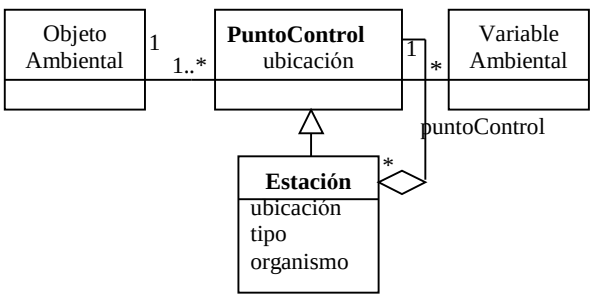


Figura 3. Metaclase PuntoControl

Medición

Se utiliza el modelo conceptual propuesto en [7], el cual adapta el patrón *Measurement* [19] que permite registrar información cuantitativa. Se combina además con una adaptación del patrón *Environmental Quality Parameters* [16] desarrollado para sistemas ambientales, tal como se muestra en la Figura 4.

Se define la metaclase Medición ambiental para denotar aquellos objetos en los cuales se almacena una serie de tiempo asociada a una variable física ambiental y se propone la utilización del estereotipo <<envMeasurement>>. Se define la restricción correspondiente al rango admisible para las mediciones de variables ambientales.

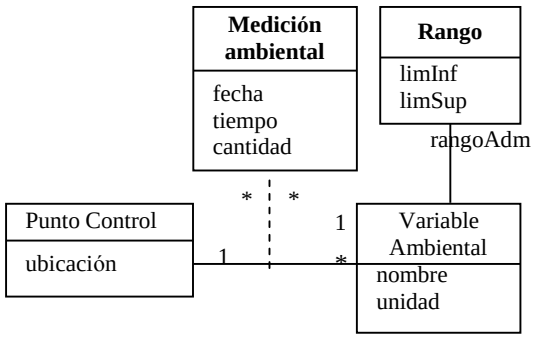


Figura 4. Metaclase MedicionAmbiental

Context MediciónAmbiental inv:
self.cantidad >=
self.variableAmbiental.rangoAdm.limInf AND
self.cantidad <=
self.variableAmbiental.rangoAdm.limSup

Mediciones calculadas

Algunas variables ambientales no pueden ser medidas directamente por medio de un sensor o instrumento. Existen métodos de cálculo para calcular el valor deseado a partir de algunas mediciones iniciales. Un protocolo de Mediciones Calculadas representa un cálculo hecho sobre mediciones presentes en el dominio. El resultado de un cálculo se trata como un objeto y conoce qué cómputos lo causaron. Los protocolos de mediciones calculadas incluyen la fórmula por la cual fueron calculados [19] como se muestra en la Figura 5.

Se introduce el estereotipo <<calcMeasurement>> para representar la metaclase MediciónCalculada en la definición del Perfil y el estereotipo <<protocolo>>.

Una restricción importante a considerar es que una medición calculada necesita de la existencia previa de mediciones de la variable vinculada a su cálculo y un protocolo asociado.

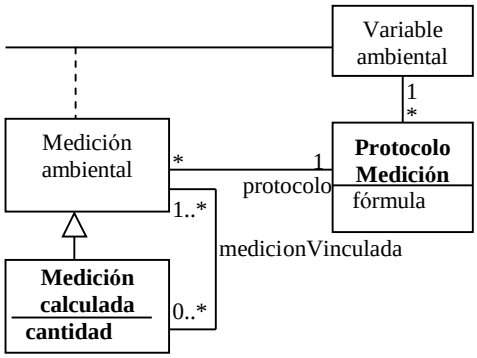


Figura 5. Metaclase MedicionCalculada

Se define la restricción para las mediciones calculadas según un protocolo de medición de la siguiente forma:

Context MedicionCalculada inv:
self.medicionVinculada -> size()> = 1 and
self.medicionVinculada.protocolo->size() = 1
self.cantidad=
self.medicionVinculada,protocolo.formula->()

Fenómeno Ambiental

El estado de las variables ambientales determina la ocurrencia de fenómenos de los cuales normalmente se registra diferente información. Estos fenómenos, se corresponden con observaciones categóricas en el Punto de Control, por ejemplo crecidas en ríos. Se introduce la metaclase Fenómeno ambiental, para la cual se utiliza el estereotipo <<envFenomeno>> y la metaclase asociativa Observación.

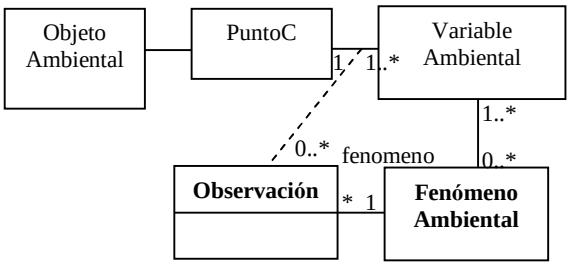


Figura 6. Metaclase FenomenoAmbiental

La restricción de existencia puede observarse en la multiplicidad de la asociación entre las metaclases VariableAmbiental y FenómenoAmbiental tal como se presenta en la Figura 6.

Metamodelo

En la Fugura 7 se define el metamodelo para las aplicaciones de monitoreo ambiental a modelar con un Perfil UML 2.0.

Para mayor claridad, en el metamodelo sólo se incluyen las clases que serán extendidas mediante estereotipos y las principales restricciones de multiplicidad.

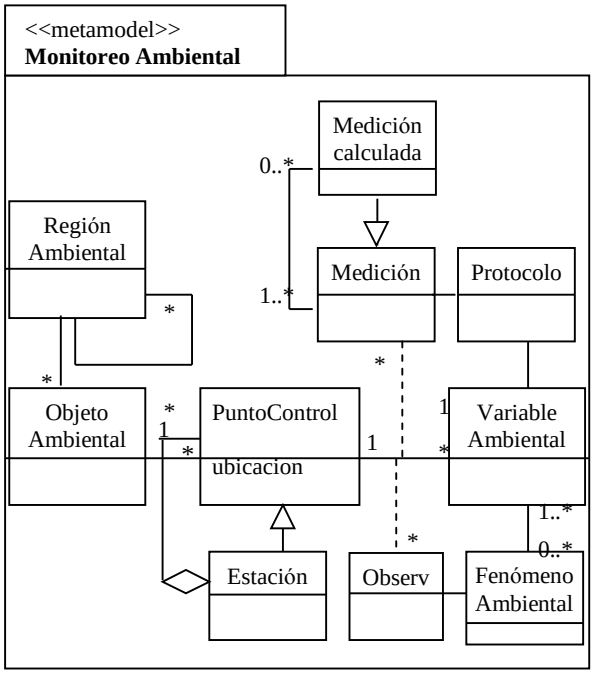


Figura 7. Metamodelo para aplicaciones SMA

3. Perfil UML 2.0 para SMA

Un Perfil se define en un paquete UML, estereotipado «profile», que extiende a un metamodelo o a otro Perfil. En UML 2.0 un valor solo puede ser representado como un atributo definido en un estereotipo, resultando así que cada elemento del modelo debe ser extendido por un estereotipo en orden a utilizar valores etiquetados. Los estereotipos son metaclases específicas y los valores etiquetados son metaatributos estándar [2].

A partir del metamodelo definido en la sección anterior, el Perfil UML 2.0 estará descripto como un paquete estereotipado <<Profile>>. El Perfil propuesto para el subdominio SMA en la Figura 8, define estereotipos correspondientes a las clases y asociaciones del metamodelo, así como las metaclases de UML sobre las que pueden aplicarse tales estereotipos. Las metaclases corresponden al metamodelo a ser extendido, es decir al de UML. Se agregan además los valores etiquetados o meta-atributos [4].

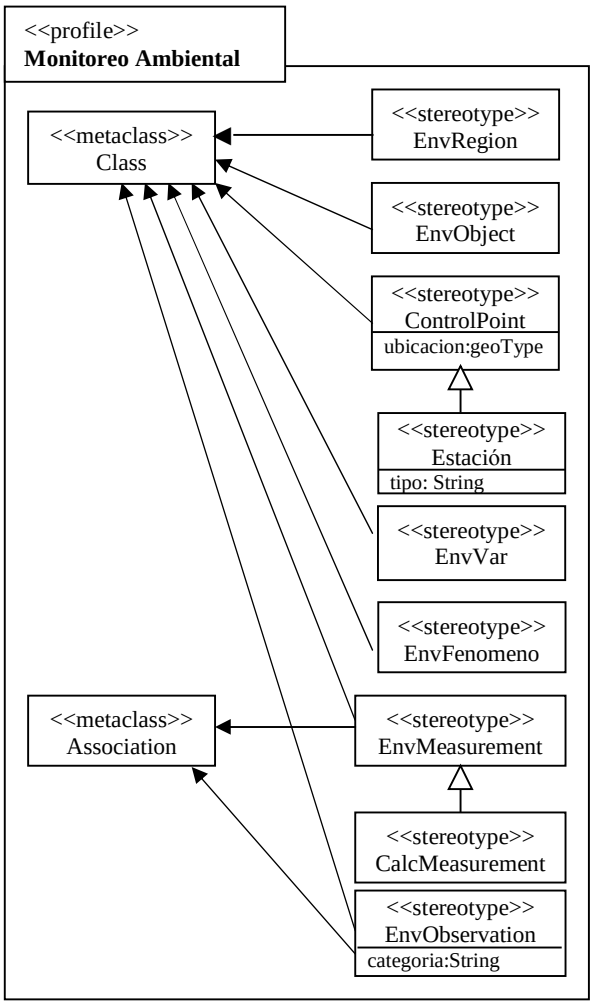


Figura 8. Perfil UML 2.0 para aplicaciones SMA

4. Caso de Estudio: Sección de Monitoreo de aguas

Se describe la extensión introduciendo un caso de estudio típico, correspondiente al monitoreo de aguas. En estas aplicaciones, usualmente se registran en distintos puntos de control y a intervalos regulares (o no) de tiempo, en forma automática y/o manual, distintas variables físicas asociadas al comportamiento ambiental de ríos, lagos, etc. De esta forma, mediante

mediciones y observaciones asociadas, puede llevarse le control y registro de los distintos fenómenos ambiéntales vinculados a los objetos físicos (contaminación orgánica, arrastre de sedimentos, etc.).

La Figura 9 muestra una aplicación que hace uso de dos perfiles, el definido en el presente trabajo y el Perfil para aplicaciones Geográficas [5].

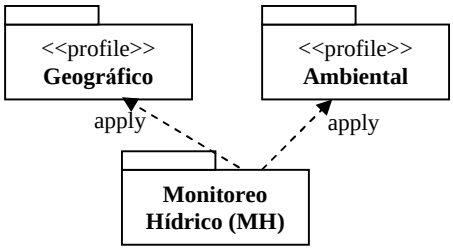


Figura 9. Uso de perfiles en una aplicación MH

A los efectos de mostrar la aplicabilidad del Perfil definido, la Figura 10 presenta el Diagrama de clases correspondiente a una aplicación de Monitoreo Hídrico. En distintas secciones de un río a través de una estación automática, se registran mediciones de nivel en forma horaria, lo cual permite el cálculo y registro del caudal. Se realizan asimismo observaciones que permiten caracterizar distintos fenómenos hídricos.

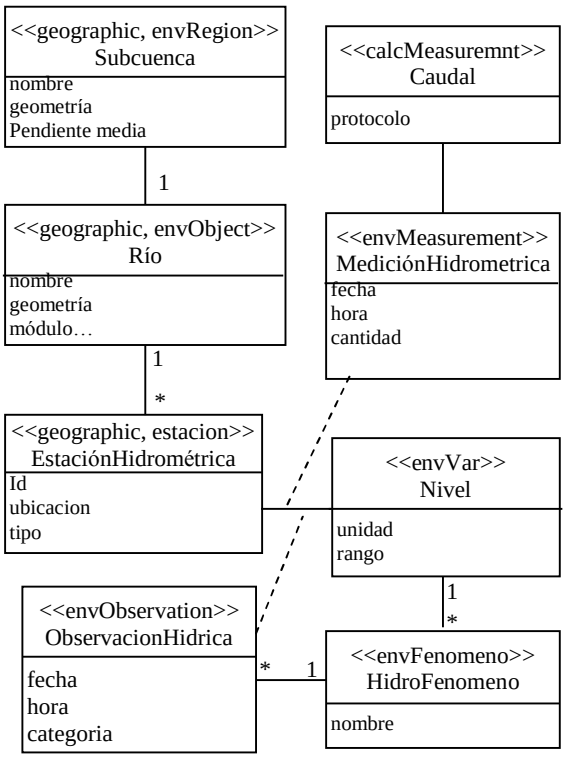


Figura 10. Diagrama de clases de la aplicación MH

El uso de las extensiones definidas en el Perfil permite una representación clara y sencilla del problema, eliminando la necesidad de utilizar modelos con complejas relaciones de herencia entre objetos ambientales, variables y fenómenos físicos, tipos de estaciones, etc., usualmente presentados en la representación de estas aplicaciones.

5. Conclusiones

En el presente trabajo se presenta un Perfil UML 2.0 para representar los principales aspectos y conceptos vinculados al modelado de las aplicaciones del subdominio de monitoreo ambiental. Esta extensión de UML contiene los estereotipos y restricciones necesarios para el modelado específico de estas aplicaciones. La definición de este Perfil sobre la base de los conceptos del dominio definidos en un metamodelo previamente analizado considerando sus principales restricciones expresadas en el lenguaje de especificación OCL, permite garantizar que las extensiones definidas se ajustan perfectamente a los requerimientos específicos de aplicaciones de SMA.

En futuros trabajos se espera refinar este Perfil agregando nuevas extensiones y restricciones y extender el perfil a otros subdominios del dominio de aplicaciones ambientales.

6. Referencias

[1] N. Moreno, P. Fraternalli, A. Vallecillo, "A UML 2.0 Profile for WebML Modeling", *ACM International Conference Proceeding Series; Vol. 155. Workshop proceedings of the sixth international conference on Web engineering ICWE '06* Publisher: ACM Press ISBN:1-59593-435-9, 2006

[2] OMG (Object Management Group) Documents "UML Superstructure Specification, v2.0" formal/05-07-04 "UML Infrastructure Specification, v2.0" formal/05-07-05 "OCL Specification, v2.0". In *UML® Resource Page*. http://www.uml.org/#UML2.0, 2005.

[3] A. Urciuolo, R. Iturraspe, A. Parsón, "Conceptual Microarchitectures for Hydrologic Simulation Models", En: *CLEI Electronic Journal*, ISSN 0717-5000, Cecilia Bastarrica ed. http://www.clei.cl. Vol 7, 1, Jun 2004, paper 6, 18 pp.

[4] Fuentes L., Vallecillo A., "An Introduction to UML Profiles", *UPGRADE*, CEPIS (Council of European Professional Informatics Societies) by NOVÁTICA, Vol. V, Nº 2, Abril 2004, pp 6-13.

[5] Kang M., Pinet F., Schneider M., Chanet J., Vigier F., "How to design geographic databases? Specific UML Profile and Spatial OCL applied to Wireless Ad-Hoc Networks", *7th*

European Conference on Geographic Information Science (AGILE 2004), Heraklion, Greece, April 2004.

[6] G. Miller, "What´s new in UML 2.0" *A Borland White Paper, Borland, 2003*

[7] A. Urciuolo, R. Iturraspe, "Conceptual Patterns for Water Resources Information Systems", En: *Journal of Computer Science and Technology* Vol. 3 - No. 1 - April 2003 - ISSN: 1666-6038, pp 20-26.

[8] S. Flake, *Temporal OCL Extensions for Specification of Real Time Constraints,* Workshop Specification and Validation of UML models for Real Time and Embedded Systems (SVERTS'03), San Francisco, CA, USA, 2003

[9] H. Baumeister, N. Koch P. Kosiuczenko and M.Wirsing "Extending Activity Diagrams to Model Mobile Systems." *In Proceedings of International Conference NetObjectDay (*NODe) '02, LNCS 2591 Springer, Germany, Oct 2002, pp. 278-293. *Lecture notes in computer science.* ISSN 0302-9743, 2003.

[10] F. Pinet and A. Lbath , "Semantics of Stereotypes for Type Specification in UML: Theory and Practice", *Book Series: Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, ISSN 0302-9743, Vol 2224, 2001, pp 339-353.

[11] M. Clauss, "Generic Modeling using UML extensions for Variability". *In Workshop on Workshop on Domain-specifc Visual Languages*, OOPSLA 2001, pages 11{18, Tampa,Florida, June 2001.

[12] J. Brodeur, Y. Bedard, M. Proulx, "Modelling Geospatial Databases using UML-based Repositories Aligned with International Standards in Geomatics". In: *Proc. of the Int. Symposium on Geographic Information System.* ACM Press. USA (2000) 39-46

[13] C. Parent S. Spaccapietra E. Zimányi, *Spatio-temporal conceptual models: Data structures + space + time*. In C. B. Medeiros, editor, Proc. ACM-GIS, pages 26--33, 1999.

[14] J. Lisboa, C. Iochpe, "Specifying analysis patterns for geographic databases on the basis of a conceptual framework". *In: Geographic Information Systems. Proceedings of the 7th ACM international symposium on Advances in geographic information systems* ACM Press New York, NY, USA ISBN:1-58113-235-2, 1999, pp 7-13.

[15] OMG (Object Management Group), Analysis and Design Platform Task Force, *"Requirements for UML Profiles, version 1.0"*, OMG document ad/99-12-32, Dec 1999.

[16] J. Lisboa F., Cirano Iochpe & K. Beard, "Applying Analysis Patterns in the GIS Domain", In: *Proceedings of SIRC`98*, Dunedin, NZ, 1998.

[17] Günther, O. *Environmental Information Systems*, Springer-Verlag, Berlín, Germany, 1998

[18] Green D., Klomp N. *Environmental informatics - a new paradigm for coping with complexity in nature.* Complexity International Vol 6, 1998.

[19] M. Fowler, *Analysis patterns: reusable object models*. Menlo Park: Addison Wesley Longman, 1997.

Una abstracción posible del Toyotismo subtensa en un Modelo Concurrente de Ciclo de Vida de software

Alejandro Estayno, Marcelo Estayno, Alicia Mon
Grupo de Ingeniería de Software G.I.S. – Universidad Nacional de La Matanza.
Buenos Aires, Argentina
{estayno, mestayno, aliciamon}@fibertel.com.ar

Abstract

The modeling of the Software Process is a guideline for the organization of the activities which involve all the development stages for the resolution of the problems that the software development presents and generates as it evolves. The modeling of the product life cycle becomes more and more complex as the technological and methodological development and the organization of the work becomes more complex.

The purpose of the Modelo Concurrente de Ciclo de Vida-1205 hereby explained is to include in the software development the Total Quality and Quality Management concepts based on the detention of the process in case of failures and through the continuous improvement. These are key elements introduced by the Toyota Model for the organization of industrial processes.

Resumen

La modelización del Proceso Software constituye un marco de referencia para la organización de las actividades que involucran todas las etapas del desarrollo, tendente a la resolución de problemas que el propio desarrollo de software plantea y genera a medida que evoluciona. La modelización del ciclo de vida del producto software se hace mas compleja a medida que el desarrollo tecnológico, metodológico y, fundamentalmente la organización del trabajo aumenta su complejidad.

En el Modelo Concurrente de Ciclo de Vida (MCCV-1205) de software que se expone en el presente artículo, se propone incorporar al desarrollo de software los conceptos de Calidad Total o Quality Management basados en la detención del proceso ante la presencia de fallos y a través de la mejora continua, elementos claves introducidos por el Modelo Toyotista de organización de procesos industriales.

Palabras Claves: Proceso Software; Modelos de Ciclo de Vida; Calidad Total

1. Introducción

La Ingeniería del Software, al igual que otras ingenierías más tradicionales, como la ingeniería eléctrica o industrial, se divide en dos grandes áreas de actividad: la ingeniería del producto y la ingeniería del proceso. Hasta la década de los 80, el área de actividad predominante fue la ingeniería del producto, debido principalmente a dos razones: la juventud de la disciplina, y el relativo desconocimiento de cómo desarrollar, desde el punto de vista puramente tecnológico, un software efectivo, fiable y de bajo costo.

El área de estudio sobre proceso software se reconoce actualmente como un factor crucial, ya que su propósito es proveer el conjunto de actividades que permiten, a partir de la expresión de una necesidad, realizada por un cliente o usuario, derivar un sistema software que satisfaga dicha necesidad. La importancia del proceso software en el contexto de la ingeniería del software está respaldada por el gran número de trabajos de investigación que se han realizado hasta la fecha. Los resultados de dichos trabajos de investigación han sido numerosos Estándares de Proceso, como el IEEE1074 [1] o el ISO/IEC 12207[2] [3], Modelos de Proceso como el Proceso Unificado de Desarrollo Software [15] Modelos de Madurez como el ISO 15504 [4] o el CMMI [5] y los diferentes Modelos de Ciclos de Vida de Software [6] que representan las transformaciones del producto software a medida que se va desarrollando a través del proceso.

El presente artículo propone una posible abstracción del proceso software y del ciclo de vida del producto, mediante la incorporación del Modelo Concurrente de Ciclo de Vida software (MCCV-1205) [7] y la aplicación de criterios específicos de organización de la producción provistos por el Modelo de Proceso industrial Toyotista, de modo tal que permita incorporar elementos que puedan analizarse en el entorno y las especificidades del desarrollo del producto software en cuanto a las formas de organización del trabajo.

2. El Modelo Toyotista

El modelo industrial japonés desarrollado y aplicado por Toyota, conocido habitualmente como *modelo toyotista* [8], introduce una serie de innovaciones en las formas organizacionales de la industria basadas en la eliminación de tiempos muertos, la desaparición del stock de productos por medio de la implementación del j*ust in time*, produciendo exclusivamente sobre pedido. A su vez, desarrolla las técnicas innovadoras del Total Quality Management (TQM) [9], basadas en la detención del proceso de producción ante la presencia de fallos y la implementación de la *mejora continua*, haciendo que el producto fluya a instancias del cliente intentando tender a la producción con "cero defectos".

La implementación de estos principios del Modelo están determinados por la *pluriespecialización* de los operadores de planta, es decir, en la ampliación de sus conocimientos y su relación participativa en la toma de decisiones y en todas las instancias de la producción, así como la especificación de principios y métodos iterativos, auto-organizativos e interdependientes en un patrón de ciclos de corta duración.

En este sentido, la producción requiere de una organización del trabajo ágil, capaz de mantener un alto nivel de adaptabilidad y flexibilidad en el empleo del trabajo y las capacidades.

Las principales innovaciones que aporta el modelo organizativo japonés, destaca las siguientes características [8]:

2.1. Técnicas de planificación

Una revolución en las técnicas de planificación y optimización de la puesta en marcha de la producción, que consiste en una inversión de las reglas tradicionales, en lugar de que la fabricación se desarrolle "en cadena" de arriba hacia abajo, se hace de abajo hacia arriba, partiendo de los pedidos y de los productos ya vendidos.

Como mencionamos anteriormente un ejemplo representativo de este tipo de organización se implementó en la producción automotriz, comenzando su línea de planificación de la producción por el último puesto de ensamblaje en la organización, que es el punto más próximo al producto terminado. A partir de los productos vendidos o entregados, se inicia un pedido de reposición sobre la cantidad exactamente vendida, sin necesidad de generar un almacenamiento de una cantidad de productos a la espera de ser entregados.

2.2. Flujo de información

Esta inversión en el "sentido" general de las instrucciones de producción, requiere de una fuente de información precisa y ajustada a la necesidad de pro-

ducción exacta en cada puesto. La clave del método consiste en establecer paralelamente al desarrollo de los flujos reales de producción (de arriba a abajo), un flujo de información invertido de abajo hacia arriba, emitiendo cada puesto corriente abajo una instrucción destinada al puesto corriente arriba inmediatamente anterior. Esta instrucción consiste en el pedido de la cantidad y la especificación exacta de las unidades necesarias al puesto corriente arriba para ejecutar su propio pedido.

Desde abajo, la serie de pedidos va de puesto en puesto y remonta corriente arriba, de tal manera que en un momento dado, en el departamento que se considere sólo hay en producción la cantidad de unidades exactamente necesaria, cumpliéndose el principio de "*cero existencias*", solo se está produciendo lo estrictamente solicitado.

2.3. Planificación

Esta modificación en la información permite descentralizar una parte de las tareas de Planificación, especializado y ajeno a los propios productores y confiar la responsabilidad de ellas a los jefes de cada equipo.

2.4. Autonomía en cada puesto

Todo el sistema de circulación de la información se lleva a cabo mediante un doble movimiento, de inscripción de pedido hacia arriba, y de productos ya producidos o desarrollados a partir de los pedidos, hacia abajo. Esto genera la necesidad de la planificación en cada puesto de trabajo, asegurando la autonomía en la toma de decisiones y el conocimiento de los trabajadores en cada puesto.

Estos tres elementos indican claramente que la innovación es sólo de organización y conceptual, sin que intervenga lo tecnológico y que responde a una organización con una inmediata capacidad de respuesta al mercado que aporta producción y montaje modular siguiendo de cerca estrictamente el flujo de los pedidos en cantidad y calidad, permitiendo de este modo, obtener una variedad de productos sobre la base de componentes elementales capaces de adaptarse a las diferentes combinaciones de productos pedidos.

2.5. Gestión de la Calidad en los propios puestos de desarrollo

La integración de las tareas de Control de Calidad de los productos a las tareas de desarrollo, permite, conjuntamente, reasociar las tareas de ejecución, programación o control de calidad, incorporando los conocimientos dados en la pluriespecialización y polivalencia de los profesionales. La reintroducción de estas tareas de Control de Calidad a los puestos de trabajo tiene por objeto que los propios puestos de desarrollo se hagan cargo de la calidad de los productos. En este sentido, la pluriespecialización de los operadores es lo que hace posible que estos se encarguen de la gestión de calidad y de algunas dimensiones de la planificación. Así también, la ejecución de las tareas variadas (fabricación, reparación, control de calidad y planificación) alimenta y enriquece permanentemente la "polivalencia" y los conocimientos prácticos de los operadores, generando un proceso de "aprendizaje dinámico".

El método japonés de linealización consiste en concebir instalaciones y una cierta distribución del espacio físico de forma tal que permita la movilización de los trabajadores pluriespecializados, y el cálculo permanente de los estándares de operación que se les asigna a cada uno. En su conjunto, la linealización constituye un sistema de recomendaciones prácticas basadas en una concepción de la producción ampliamente renovada, que se centra en movilizar los recursos de "tiempos y movimientos" en organizaciones e implantaciones flexibles.

El dispositivo destinado al mejoramiento de la calidad de los productos, introduciendo en el nivel de los propios procesos operatorios dispositivos de llamado que tienen por objeto prevenir el error, hacerlo casi imposible, generando el paro de una máquina o un proceso en cuanto se presenta un riesgo de defecto de fabricación o de desarrollo. Se trata de dispositivos que pueden adaptarse a los equipos y a las herramientas tendiendo a "cero defectos" [8].

Este tratamiento de la calidad implica quedarse con la manera de ejecutar una tarea cualquiera, que presente la mayor garantía en cuanto a la calidad del producto, en vez de buscar la rapidez de ejecución. Sobre esta base los japoneses han elaborado progresivamente un impresionante conjunto de herramientas de Gestión de Calidad que culminan en las diferentes técnicas llamadas de "Calidad Total" [10].

Procedimientos que permiten hacer literalmente "visible" el desarrollo del proceso de producción al permitir una visualización de cada uno de los acontecimientos susceptibles de producirse como puede ser el exceso o insuficiencia de existencias de "requisitos" con relación a los pedidos, interrupción o disminución de la velocidad del flujo. Esta disponibilidad física y práctica, lo que genera básicamente es incitar a los trabajadores a que no duden en detener el proceso de producción, es el mejor medio para asegurarse de que se hará todo para eliminar prontamente las anomalías. Este control visual y la incitación al autocontrol son características muy claras del método japonés.

3. Modelo de Ciclo de Vida Concurrente MCCV-1205

Dentro de las actividades asociadas a las fases de un ciclo de vida de software no se puede pasar de un estado a otro, ni siquiera ejecutar un proceso por simple y mínimo que fuera dentro del Ciclo de Vida sin algún tipo de conocimiento. El proceso de adquisición del conocimiento no es secuencial y menos aún lineal. El planteo básico es el siguiente: cada porción del conocimiento necesaria puede ser transmitido de un proceso -subproceso- a otro, para lograr pasar de un estado -intermedio- a otro. El ciclo de vida de transformación del producto software no se expresa como un proceso secuencial, es un proceso concurrente y no hay posibilidades de determinación para las coordenadas temporales de los pasajes de conocimiento.

En este sentido y para ejemplificar con una parte del proceso, si alguna de las subetapas de la etapa de análisis comienza, es porque se tiene el conocimiento, concreto y necesario para darle inicio. El conocimiento es portado, por ejemplo, por un entregable, como input del proceso asociado a la subetapa. Este entregable también representa el momento en que están dadas las condiciones para comenzar a ejecutar el proceso, por medio del cual se cambia de estado, progresando en la dinámica del ciclo de vida del producto.

3.1. La concurrencia en el proceso

En un proceso de construcción de la solución en términos de los recorridos hacia la solución definitiva del problema, su dinámica no luce como un proceso de derrotero continuo. Haciendo una analogía con el proceso mental disparado ante un problema emergente a solucionar. Al instante de conocer el problema que se resolverá con la implementación de una solución concretada en un sistema software, se establecen una confluencia caótica de escenarios y actos mentales.

Una alternativa posible, para la resolución del problema, para alguien con cierta experiencia en el desarrollo de aplicaciones software podría ser:

- primera versión psíquica de la interfase software
- conjunto mínimo de funcionalidades básicas
- ciertas funcionalidades esenciales
- algunos problemas de las funcionalidades
- algunas soluciones a un subconjunto de los problemas anteriores
- algún integrante del team a quien consultar problemas a resolver
- analogías con sistemas conocidos
- posibles reutilizaciones de soluciones implementadas
- primera visión de la estrategia tecnológica
- universo de usuarios
- imagen de un layout posible del team project
- alguna circunstancia en el escenario de desarrollo de la solución
- estimación bruta del tiempo
- ...

Aunque en términos clásicos se esté en medio de un estadio típico de captura de requerimientos, concurrentemente se han establecido conexiones y se ha logrado un avance en el recorrido hacia la solución del problema. En estos caminos alternativos, se han disparado procesos que anticipan ciertas fases del ciclo de vida de desarrollo de un producto software como análisis, prototipado, diseño, test y codificación. Aún más, este proceso se hace constante y tiene como único límite la finalización del ciclo de vida.

Dejando a un lado, por el momento, la concurrencia de avances sobre las fases del ciclo de vida, surge la cuestión sobre cuáles son las condiciones para que cada uno de ellos avance o se detenga. Es por ello que, la incorporación del mecanismo de semáforos, permite incorporar al modelo de ciclo de vida propuesto, una herramienta teórica para facilitar la detención o continuación del proceso.

3.2. Los semáforos de Dijkstra

Los *semáforos,* concebidos por Edsger Wybe Dijkstra [11] [12], constituyen una herramienta de sincronización que restringe o permite el acceso a recursos, que en el caso del MCCV-1205, los recursos o entregables están conformados por el conocimiento.

De acuerdo a las definiciones presentadas en Operating System Concepts [13], un semáforo **S** es una variable entera a la que, una vez que se le ha asignado un valor inicial, sólo puede accederse a través de dos operaciones atómicas estándar: espera (wait) y señal (signal). Estas operaciones se llamaban originalmente **P** (para espera; del holandés *proberen*, probar) y **V**

(para señal; del holandés *verhogen*, incrementar).

El valor de una variable del tipo semáforo es el número de unidades del recurso (en nuestro caso conocimiento/s) que está/n disponible/s. La operación P detiene la ejecución hasta que hay un recurso disponible, en cuyo caso lo reclama inmediatamente y sigue su ejecución normal. V es la operación inversa; hace disponible un recurso común, y permite que uno de los procesos suspendidos por la falta de ese recurso, prosiga con su ejecución. La operación *Inicia* se utiliza para inicializar el semáforo antes de que se hagan peticiones sobre él.

Las definiciones clásicas de espera y señal son:

Espera(S):

```
while S ≤ 0 do nada;
S:= S - 1;
```

Señal(S):

```
S:= S + 1;
```

Las modificaciones del valor entero de los semáforos en las operaciones Espera y Señal se deben ejecutar de forma indivisible, es decir, mientras un proceso modifica el valor del semáforo, ningún otro proceso puede modificar simultáneamente el valor de ese mismo semáforo. En el caso de Espera(S) la prueba del valor entero de S ($S \leq 0$) y su posible modificación ($S := S - 1$), también deben ejecutarse sin interrupción.

3.3. Los semáforos y el modelo

En el MCCV-1205 cada fase de un ciclo de vida se comporta como un dispositivo receptor y transmisor de conocimiento sobre *cómo* debe/n seguir otra/s fase/s. De este modo, cada instancia del producto, desarrollada en una fase del ciclo de vida, podría pasar a otra fase del desarrollo, cualquiera que sea, solo si tiene una aceptación de que todo lo necesario para esa parte ha sido completada y si la fase que la requiera para iniciar su ejecución cuenta con los conocimientos necesarios para comenzar dicha instancia, y cada fase continúe trabajando en forma concurrente con otra en la misma parte del producto.

La sucesión de instancias está determinada por las condiciones para que cada proceso, avance o se detenga. Para ello, el MCCV-1205 incorpora el mecanismo de semáforos, para instanciar los permisos que cada fase tiene para entregar y/o para recibir una parte del producto.

Los semáforos, en este caso resignificados en una herramienta conceptual de sincronización, son utilizados para modelar el flujo en el tiempo de las actividades asociadas a cada fase, que cooperan entre sí transfiriéndose el conocimiento necesario para proseguir. Cada actividad asociada a cada fase de un ciclo de vida se comporta como un dispositivo receptor y transmisor de conocimiento sobre cómo debe /n seguir otra /s fase /s.

Por ejemplo, si suponemos un conjunto extremadamente clásico de fases: Requerimientos, Análisis, Diseño, Implementación y Pruebas, bajo las concepciones de un modelo secuencial clásico, la tecnología a aplicar en el desarrollo de una solución software es una decisión de diseño. Pero si en la etapa de requerimientos, con los primeros conceptos, más allá de las técnicas empleadas para la captura de requisitos, se obtiene información básica sobre la tecnología del sistema de gestión de base de datos implantada en las instalaciones sobre las que se va a realizar el deployment del sistema a desarrollar, estamos en presencia de una posible concurrencia en la evolución de las fases.

Este fenómeno, en términos de modelización, puede ser asociado a un arreglo unidimensional KT (Knowledge Transferor), compuesto de semáforos KT_f donde $f \in \mathbf{N}$ en el rango [1..n], siendo n la cantidad de fases distinguibles en el C de V y f un identificador de una de sus fases.

El valor de cada semáforo es 0 (cero) si la fase no tiene los *conocimientos* suficientes para continuar y su valor varía de 0 a n dependiendo de la *carga de conocimiento* transferida por la fase *transmisora,* permitiéndole avanzar en la solución. En principio, el vector está inicializado a 0 -al tratarse de un problema desconocido, no hay conocimiento básico para transferir-.

Una abstracción posible sobre este modelo puede expresarse en términos del siguiente *pseudocódigo*:

```
/** Declaración e iniciación de un arreglo global de semáforos de transferencia de conocimientos a fases del ciclo de vida **/
```

```
var
KT : array [1.. n] of semaphores

Análisis := 1;
Diseño := 2;
.....
Pruebas := n;

Begin
        for i := 1 to n do
                inicia(KT[i], 0)


/** Cómo cada fase es sincronizada dentro de la dinámica del C de V **/
Requisitos : : Fase de C de V
Begin
  while .t. begin
        P(KT [Requisitos]);
                                        /** si tiene conocimientos, o sea, si KT[Requisitos] > 0, avanza, si no espera por
                                        conocimiento transferido desde alguna otra fase **/

        ... evolución de la fase ...

        case conocimientos para transferir a Análisis
                V (KT[Análisis]);
                                        /** si tiene conocimientos para transferir a Análisis, ejecuta un V sobre el semáforo
                                        de avance de Análisis habilitando a la fase a continuar avanzando en su evolución **/

        case conocimientos para transferir a Diseño
                V (KT[Diseño]);
                                        /** Ídem anterior, pero con respecto a Diseño **/
        .....
        case conocimientos para transferir a Pruebas
                V (KT[Pruebas]);

                                        /** Ídem anterior, pero con respecto a Pruebas **/
  end.
end.
```

La sucesión de fases en forma concurrente del Modelo Concurrente de Ciclo de Vida, podría representarse gráficamente tal como se presenta en la siguiente figura:

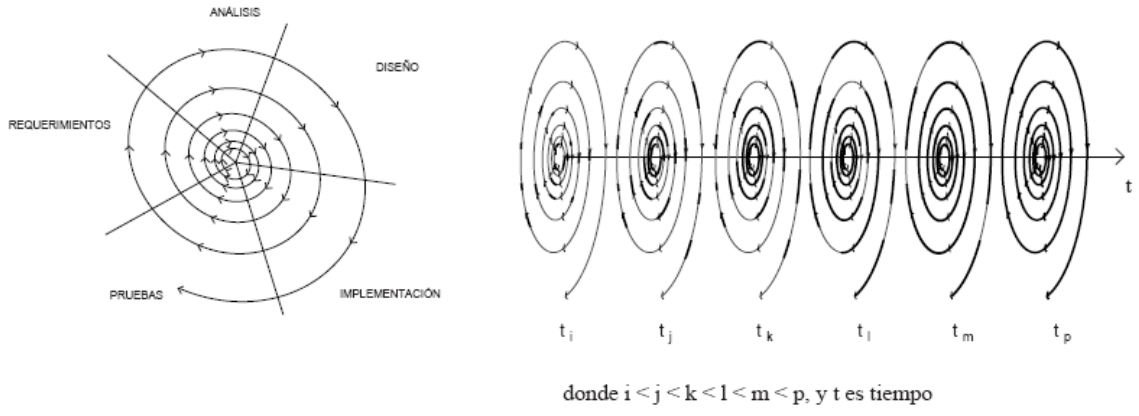


Figura 1 – MCCV-1205

En la gráfica, se ha tomado como modelo de ejemplo, la espiral de Boehm [14]. Si consideramos a su esquema gráfico como el lugar abstracto donde transcurren los eventos de la dinámica asociada al ciclo

de vida, se podrán observar las "instantáneas" para cada t_i, t_j, ..., t_p, representada por cada una de las espirales que se suceden en la figura, atravesadas por el eje del tiempo. Se observa, en cada espiral o instante del tiempo, como se va "completando" el ciclo de vida del producto software de manera indeterminística.

Este proceso solo corresponde a ese lapso de tiempo, comprendido entre t_i y t_p, y para un caso en particular. De lo que se desprende que, aunque probabilísticamente posible, a los fines prácticos es imposible encontrar secuencias de evolución iguales.

Dado que el modelo es concurrente y las actividades de cada fase son disparadas de manera no determinística, no se puede predecir cómo se ordenarán las fases del ciclo de vida, lo que revela un nuevo planteo.

- Toda fase existe desde el instante $T_i = 0$;
- Se instancian de manera no determinística, de acuerdo con una función de correspondencia estocástica, con una distribución de probabilidades, generando un posible derrotero del ciclo de vida;
- Sus finalizaciones se desencadenan ordenadamente, de acuerdo con el modelo clásico, a pesar de que el momento de nacimiento de cada fase tiene un grado no determinado de incertidumbre, el orden de finalización está claramente determinado. La probabilidad P de que la fase de requerimientos finalice antes de la fase de implementación es 1, de otro modo, la fase de requerimientos incluiría actividades que no tendrían impacto en la implementación, de lo que se desprendería un cuestionamiento sobre el sentido de estas actividades.

Desde un punto de vista general se puede abordar el tema de la paralelización de las tareas desde dos visiones:

1. La posibilidad llevar a cabo las tareas que deben ser explotadas en paralelo mientras sus dependencias temporales lo permitan.
2. La posibilidad llevar a cabo tareas que se presentan de modo indeterminístico y concurrente.

Estas dos visiones confluyen en un mismo resultado, pero de acuerdo al abordaje que se realice, se planteará la solución.

Incorporando al desarrollo de software los conceptos del Total Quality Management basadas en la detección del proceso de producción ante la presencia de fallos y a través de la mejora continua, suponemos que cada instancia del producto, desarrollada en una fase del MCCV-1205, puede pasar a otra fase del desarrollo, cualquiera que sea, solo si tiene una aceptación de que todo lo necesario para esa parte ha sido completada y si la fase que la requiera para iniciar su ejecución cuenta con los conocimientos necesarios para comenzar dicha instancia, y cada fase continúe trabajando en forma concurrente con otra en la misma parte del producto.

De esta manera, surge la cuestión sobre cuáles son las condiciones para que cada producto en cada fase avance o se detenga. Para ello, el Modelo MCCV-1205 incorpora el mecanismo de semáforos, para instanciar los permisos que cada fase tiene para entregar y/o para recibir una parte del producto (conocimiento).

4. Conclusiones y trabajo futuro

En el presente artículo se ha presentado un modelo de ciclo de vida que permite la confluencia de diversos aspectos provenientes de la ingeniería de procesos del modelo toyotista y el MCCV-1205.

La intención del mismo es centrar el análisis en la transformación constante de los elementos organizativos del proceso software, de manera tal que permita introducir en el ciclo de vida del software, el concepto de detención del proceso ante la *presencia de fallos,* la aplicación de *semáforos* en el desarrollo para la detención o continuidad del proceso y la *transferencia de conocimiento*.

Es por ello que se han destacado tres factores diferenciales contenidos en los conceptos principales de los modelos presentados.

1. La paralelización y concurrencia de tareas.

2. La detención del proceso, donde la longitud[1] de las líneas de producción, son proporcionales al esfuerzo que conlleva mantener la calidad hasta la confección del entregable final.

3. La incertidumbre sobre el momento en que aparece el "error".

De acuerdo al MCCV-1205, la concurrencia de las posibilidades de progresar en las actividades que provocan los cambios de estado en el ciclo de vida del software se presentan sin ningún tipo de control sobre el flujo temporal de sus apariciones. El Modelo Toyotista incorpora las técnicas de Total Quality Management basadas fundamentalmente en la detención del proceso productivo en cada módulo ante la presencia de fallos.

[1] Usamos aquí la noción de longitud, pero ésta solo trasciende como una metáfora del tiempo que toma completar el camino crítico del cronograma de trabajo.

En la presente investigación y como línea futura de trabajo, queda por definir cuáles son las condiciones para que cada producto en cada fase avance o se detenga. Para ello, el MCCV-1205 incorpora el mecanismo de semáforos, para instanciar los permisos que cada fase tiene para entregar y/o para recibir una parte del producto, aplicados también por el toyotismo en la línea de producción. Si bien los conceptos de parar por un error (falta de conocimiento) o avanzar por la solución del error (obtención de conocimiento) resultan los elementos claves, queda aún por determinar las técnicas y/o prácticas que permitirían instanciar el punto de avance o detención en cada una de las fases.

5. Referencias

[1] *IEEE Standard for Developing Software Life Cycle Processes*, IEEE Standard 1074-1997.

[2] *ISO/IEC International Standard: Information Technology. Software Life Cycle Processes*, ISO/IEC Standard 12207-1995.

[3] *ISO/IEC International Standard: Information Technology. Software Life Cycle Processes*, Amendment 1, ISO/IEC Standard 12207-1995/Amd. 1-2002.

[4] *ISO/IEC 15504, Information Technology – Software Process Assessment*. International Organization for Standardization, International Electrotechnical Commission. 1998. http://isospice.com/standard/tr15504.htm.

[5] Carnegie Mellon, Software Engineering Institute. *Capability Maturity Model® Integration (CMMISM), Version 1.1. CMMI^{SM} for Systems Engineering, Software Engineering, Integrated Product and Process Development, and Supplier Sourcing. (CMMI-SE/SW/IPPD/SS, V1.1).* Carnegie Mellon University. 2002.

[6] L. Alexander and A. Davis, "Criteria for selecting software process models". *Proceedings of COMPSAC'91*. 521-528. 1991.

[7] Estayno, Alejandro. *El Modelo Concurrente del Ciclo de Vida - MCCV - 1205*. XII Congreso Argentino de Ciencias de la Computación. 2006.

[8] Coriat. B.: *Pensar al Revés: Trabajo y Organización en la Empresa Japonesa*. Siglo XXI, 1992.

[9] Liker J. *The Toyota Way: 14 Management Principles from the World's Greatest Manufacturer*. Mac Graw Hill. 2004

[10] Fruin. M.: *Las Fábricas del Conocimiento, la Administración del Capital Intelectual en Toshiba*. Ed. Oxford, 2000.

[11] E. W. Dijkstra, *Solution of a problem in concurrent programming control.* Communications of ACM. NuevaYork. 1965.

[12] Dijkstra E. W. *Cooperating sequential processes.* University of Texas. NuevaYork. 1968.

[13] A. Siberschatz y P. B. Galvin. *Operating System Concepts*. Addison Wesley. Massachusetts. 1998

[14] B. Boehm. "A Spiral Model of Software Development and Enhancement" *Computer*, pp. 61-78, May 1988.

[15] Jacobson, I; Booch, G; Rumbaugh, J. *El Proceso Unificado de Desarrollo de Software,* Addison Wesley,1999,

Aplicación de la tecnologia Bluetooth orientada a la integración de servicios de Internet en dispositivos móviles

Juan Guillermo Torres Hurtado
Universidad del Valle, Colombia
juan_torres@ieee.org

Álvaro Bernal Noreña
Universidad del Valle, Colombia
alvaro@univalle.edu.co

Abstract

The impact in our society of the mobile devices has allowed the development of new applications; one of these applications is Internet access service, which has become an important way of communication and organization in our life. This paper shows the design and implementation of a communication interface oriented to allow access services between mobile devices (PDA's and telephones) using Bluetooth. This model offers an economic option when the Internet service is being used through a Local Area Network (LAN)

1. Introducción

El impacto que ha generado la masificación de los dispositivos móviles en la sociedad, ha promovido el desarrollo de nuevas aplicaciones diferentes a aquellas para las que fueron inicialmente creadas, una de éstas permite prestar el servicio de acceso a Internet, que sin duda se ha convertido en un medio vital de comunicación y organización de nuestras vidas.

En éste trabajo se presenta el diseño e implementación de una interfaz de comunicación que permite acceder a servicios de Internet desde dispositivos móviles como las PDA's y teléfonos celulares, usando tecnología Bluetooth. Además éste modelo brinda una opción más económica al hacer uso de los servicios de Internet permitiendo acceder a través de una red LAN

Éste trabajo fue desarrollado dentro de un proyecto de investigación titulado "Apropiación de una tecnología para acompañantes móviles digitales", el propósito de éste proyecto es concebir y diseñar un acompañante móvil digital con una arquitectura abierta que permita tomar provecho de las actuales tendencias tecnológicas, incluyendo las comunicaciones inalámbricas .

El trabajo es organizado de la siguiente manera: en la segunda sección se explica en que consiste la tecnología Bluetooth, los diferentes perfiles utilizados y sus alcances; en la sección 3, se detalla de forma general la totalidad de la topología implementada; la sección 4, explica los procedimientos efectuados por cada dispositivo inalámbrico para el control e implementación de la red Bluetooth, además podemos observar el mecanismo como se ha conformado la red entre los diferentes dispositivos Bluetooth; la sección 5 explica la manera como se realiza la integración entre una red LAN y una red Bluetooth; finalmente la sección 6 expone el modelo utilizado por la aplicación para el acceso a Internet de un dispositivo móvil.

2. Funcionamiento de la Tecnologia Bluetooth

Bluetooth[1] es una tecnología de comunicación inalámbrica de corto alcance, inicialmente desarrollado para reemplazar las conexiones hechas por cable. Estas conexiones utilizan frecuentemente enlaces punto a punto entre dispositivos Bluetooth, no obstante las especificaciones del protocolo ofrecen soluciones para las topologías de interconexión más complejas, llamadas Piconets ó Scatternets, que provean una efectiva y eficiente comunicación sobre múltiples conexiones con tiempo de respuesta y consumo de potencia aceptables en el desarrollo de soluciones inalámbricas.

Bluetooth provee avances significativos sobre otras tecnologías de comunicación, tal como IrDA y Home RF, la debilidad de IrDA se debe a que ofrece conexiones de muy corto alcance (un metro de distancia) con línea de vista, lo cual impide la comunicación entre dispositivos que no estén visibles el uno del otro.

Dado que Bluetooth hace el uso de RF para la comunicación, no esta sujeto a éste tipo de limitación y puede conectarse con dispositivos que estén a un alcance de 10m (hasta 100m si la potencia de transmisión es incrementada), diseñados para ser de bajo costo, sin embargo el numero de conexiones activas están limitadas a siete dispositivos, con una tasa de transmisión de hasta 3Mbps para Bluetooth 2.0 + EDR.

Finalmente la principal ventaja de Bluetooth es la habilidad de manejar trasmisión de voz y datos simultáneamente, con capacidad de soportar un canal de datos asíncrono y hasta tres canales de voz síncronos, ésta habilidad combinada con conexiones de tipo **Ad Hoc,** unida a una aplicación capaz de descubrir los servicios que prestan los diferentes dispositivos de la red, la convierte en una excelente solución de comunicación para dispositivos móviles y aplicaciones de Internet.

2.1. Perfiles de conexión de la Tecnología Bluetooth

Los perfiles[2] definen las características que debe soportar el Stack de Bluetooth para poder realizar una función específica, cuando varios dispositivos provenientes de diferentes fabricantes conforman los mismos perfiles, estos tiene capacidad de interoperatividad para un servicio en particular.

Generic Acces Profile: éste perfil define el procedimiento general para descubrir dispositivos Bluetooth, control de enlace y conexión. El principal propósito de éste perfil es describir el uso de las capas inferiores (Link Control, Link Manager), además define procedimientos de seguridad que puede desempeñar las capas superiores (L2CAP, RFCOMM, OBEX), incluyendo los parámetros comunes de accesibilidad desde una interfaz de usuario.

Service Discovery Application Profile: define la búsqueda de servicios habilitados a un dispositivo Bluetooth, permitiendo conocer servicios específicos y generales de cada dispositivo.

Serial Port profile: éste perfil define como los dispositivos Bluetooth son configurados para emular una conexión de cable serial usando RFCOMM, que es un protocolo de transporte que emula el puerto RS232 entre dos dispositivos, RFCOMM es usado para transportar datos de usuario, señales de control de MODEM y comandos de configuración.

Dial up Networking Profile: define los procedimientos usados por los dispositivos tal como los Modems y teléfonos celulares, una de las posibles aplicaciones de éste modelo, consiste en el uso de los teléfonos móviles como MODEM de acceso a Internet inalámbrico.

Generic Object exchange Profile: éste perfil define el conjunto de protocolos y procedimientos para el intercambio de objetos entre dispositivos, los modos de uso son: transferencia de archivos y sincronización.

La figura 1 muestra los perfiles utilizados dentro de la red Bluetooth soportados por el perfil principal SPP.

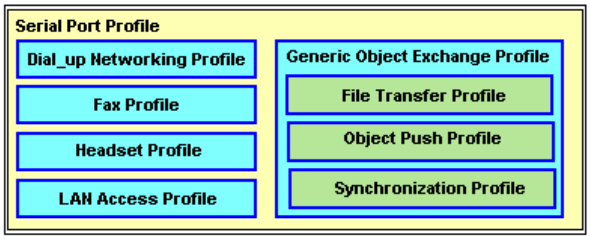


Figura 1. Perfiles y modos de uso soportados por el Perfil de puerto serial

3. Topología de Red Implementada

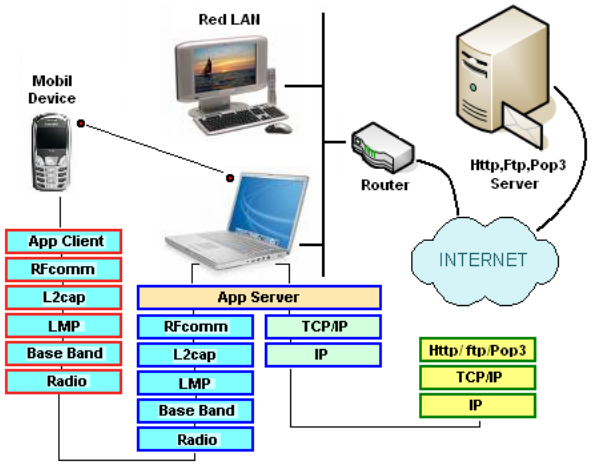


Figura 2. Topología de Red Implementada.

La topología de conexión que se ha desarrollado en éste trabajo esta constituida de tres partes principales, la primera de ellas está conformada por una red de dispositivos Bluetooth con capacidad de usar los perfiles Bluetooth GAP, OBEX, DUN y LAN Access.

La segunda etapa consta de una red LAN, a la cual se le adiciona un dispositivo Bluetooth a uno de los Host de ésta Red usando una conexión RS232, con el propósito de enlazar las dos redes, y la tercera etapa esta constituida por el enlace entre uno de los Host de la red LAN y cualquier destino al que se desee acceder a través del Internet. La figura 2 nuestra completamente la topología de red implementada.

3.1. Diseño, Control y organización de una Red tipo Ad Hoc

En ésta etapa se pretende implementar un Red Bluetooth con la capacidad de transferir información entre los diferentes dispositivos, para éste caso es indispensable saber el tipo de trasmisión que se desea aplicar, ya que podemos trasmitir datos o voz utilizando enlaces orientados o no orientados a conexión. Para la implementación de éste tipo de red se ha utilizado módulos Bluetooth comerciales, los cuales soportan las interfaces de conexión USB, UART, SPI, PCM e implementan el Stack de Bluetooth desde la capa de Banda Base hasta la capa de RFCOMM y parte de las capas SDP y OBEX.

Estos dispositivos son controlados mediante una conexión serial RS232 del Host PC en el cual corre el software que realiza el control de gestión para la transferencia de datos entre los diferentes dispositivos Bluetooth. El funcionamiento de la aplicación implementada, mostrada en la figura 3, opera de la siguiente forma.

Req_Inquiry: el primer paso consiste en descubrir cuales son los dispositivos Bluetooth que se encuentran dentro de una radio de 10/100 metros, e identificar su dirección física.

Res_MAC_Add: los dispositivos cercanos responden a éste requerimiento con su propia dirección física, dado que la dirección física no ofrece suficiente información que describa la utilidad del dispositivo se procede a obtener el nombre del dispositivo que generalmente es asignado por el propietario.

Req_DSP: seguidamente el usuario especifica con cual de los dispositivos encontrados desea establecer una conexión y además realiza una exploración del tipo de servicio que puede ofrecer dicho dispositivo; en ésta etapa se hace uso del protocolo de descubrimiento de servicio (SDP) para identificar que tipo de servicio puede ofrecer el dispositivo al cual deseamos conectarnos. Los servicios que brindan son variados tal como OBEX perfil de transferencia de archivos, manos libres, Dial up Networking ,SPP, etc. Además especifica el número del canal RFCOMM por el cual presta dicho servicio.

Autorización / PIN: Cuando se pretende acceder a un dispositivo Bluetooth por primera vez, debe autorizarse la conexión y en la mayoría de los casos se solicita el código de identificación único. Después de introducir correctamente el código, el dispositivo responde enviando los diferentes servicios que puede ofrecer.

Abrir conexión: Después que el usuario específica el tipo de servicio que desea utilizar, se adicionan los niveles superiores del Stack de protocolos necesarios para prestar dicho servicio y se establece la conexión Bluetooth. El procedimiento efectuado para el control de los dispositivos Bluetooth puede observase en la figura 3.

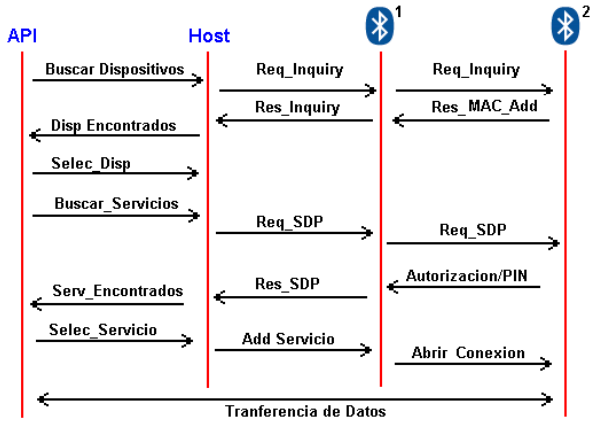


Figura 3. Procedimiento de control de los dispositivos Bluetooth

Para validar nuestra aplicación, se realizaron pruebas con un teléfono móvil Motorola L6 a fin de acceder a los diferente servicios que éste puede ofrecer, los servicios probados son los siguientes: DUN Dialing Up Network con el cual se pode manipular un teléfono móvil mediante comando AT, OBEX Object push profile server para la transmisión de fotos, juegos y música, y Hands-free, además se

hicieron pruebas con otros dispositivos Bluetooth brindando la opción de establecer transmisión de voz con un enlace orientado a conexión, y trasmisión de datos haciendo uso de perfil de trasmisión serial SPP, La figura 4 muestra el procedimiento utilizado para acceder a un teléfono móvil con conexión Bluetooth usando el perfil DUN.

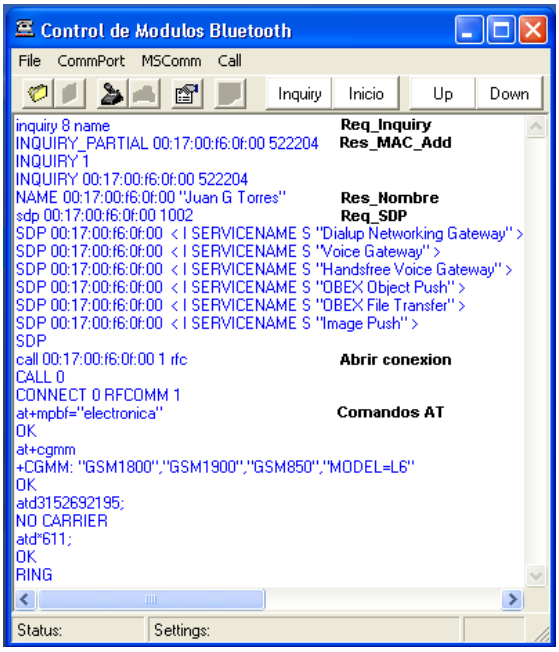


Figura 4. Procedimiento utilizado para acceder al un teléfono móvil con conexión

4. Conformación de la Piconets y Scatternets

El propósito es conformar una sencilla red Bluetooth[3], con capacidad de que al menos uno de los dispositivos pueda acceder a Internet. La primera tarea a desempeñar es conformar varios enlaces entre módulos Bluetooth, a éste tipo de conexión se le conoce como Piconets. Cuando la transferencia de información entre los módulos se hace a través de varios dispositivos que sirve de puente de comunicación entre el modulo transmisor y el modulo receptor se le conoce como Scatternet, por otro lado también se desea tener control sobre cada uno de los dispositivos que intervienen en la Pinonet ó Scatternet, esto implica que nuestra red debe realizar un procedimiento donde se cree un registro de los módulos que la componen y sea conocido por cada uno de los dispositivos.
El procedimiento que se ha propuesto para la conformación de la red se muestra a continuación:

1. Inicialmente debe haber por lo menos un dispositivo que se encargue de realizar el procedimiento INQUIRY a fin de obtener la dirección MAC de todos los dispositivos (máximo 7) que se encuentren dentro rango del alcance.

2. El modulo que ha realizado el proceso de Inquiry debe suministrar a cada uno de los dispositivos encontrados la información sobre la dirección MAC de los demás dispositivos que conforman la red. De esta manera queda conformada completamente la Piconet,

Después de realizado éste procedimiento, vemos que cada dispositivo puede compartir información dentro de la red Bluetooth.

5. Integración entre una Red Bluetooth y una Red LAN

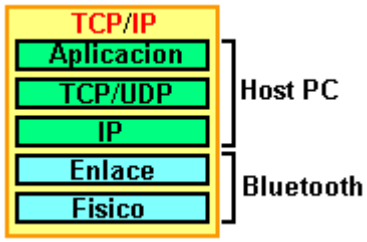


Figura 5. Integración de Protocolo Bluetooth con el Protocolo TCP/IP

En esta etapa se pretende integrar la red Bluetooth y la red LAN. Para éste propósito nuestro modelo consiste en utilizar uno de los Host PC que componen la red LAN a fin de integrarle tecnologia Bluetooth a través de uno de los puertos de conexión del PC, por simplicidad se opto por el uso del puerto serial RS 232 y se ha diseñado una aplicación que se encarga de acceder a los servicios de Internet: SMTP, POP3 y FTP, obtener la información requerida para luego ser retransmitidos a través de un canal RFCOMM hacia un dispositivo móvil, obteniendo un Host con la capacidad de interactuar con una red LAN y la red Bluetooth , y en éste caso el Host PC hará las veces de maestro dentro de la Piconet. Ahora bien, los dispositivos Bluetooth ofrecen en su Stack de protocolo hasta el nivel RFCOMM, es así que las tramas que obtenemos a través del puerto serial son del mismo tipo, no obstante para poder acceder a la red LAN debe obedecerse el protocolo TPC/IP, de esta forma la aplicación que se implementa en el Host PC se encarga de acoplar los dos protocolos (ver figura 5). El enlace Bluetooth nos ofrece los primeros dos

niveles de protocolo TCP/IP (Físico, Enlace) nuestra aplicación se encarga de interpretar los datos que obtenemos de los módulos Bluetooth, y los trasporta los niveles superiores (IP, TCP-UDP, Aplicación)

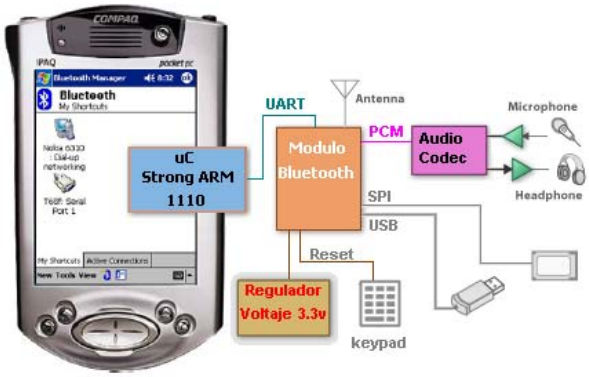


Figura 6. Diagrama de Bloques de la arquitectura hardware implementada para el acceso a Internet usando un enlace Bluetooth

En la figura 6 se observa el diagrama de bloques de la arquitectura hardware que se ha implementado a fin de proveer un enlace Bluetooth entre los dispositivos móviles y el Host PC, dado que la tecnología Bluetooth debe soportar la capa de enlace y la capa física del protocolo TCP/IP, se ha dotado de puertos de conexión UART, USB, SPI y PCM para la transmisión de Voz, a fin de soportar cualquier tipo de puerto que el usuario desee implementar.

6. Acceso a los servicios de Internet desde un dispositivo móvil.

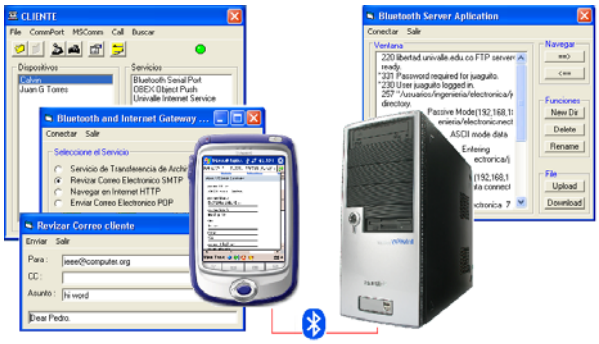


Figura 7. Interfaz Grafica de la Aplicación Cliente/Servidor orientada a los servicios de Internet

Dentro de la red Bluetooth pueden encontrarse dispositivos que deseen acceder a los servicios de Internet. La metodología que se ha adoptado [4], consiste en agregar características de conexión Bluetooth a una PDA que no posea esta forma de comunicación, la PDA utilizada para nuestro fin es una IPAQ 3600, la manera como hemos agregado el Bluetooth a la PDA es exactamente igual a como se ha agregado ésta tecnología al Host PC. Dentro de la PDA corre una aplicación ***Cliente*** que permite el acceso al usuario de los servicios de Internet tal como Ftp, Smtp y Pop3 además ésta aplicación se encarga de realizar el enlace de comunicación Bluetooth, la aplicación constantemente reenvía los datos con su respectivo puerto que identifica la aplicación y son transportados por el puerto serial de la PDA hacia el modulo Bluetooth, esta información es capturada a través de un enlace Bluetooth usando el perfil de comunicación serial y es interpretada por una aplicación ***Servidor*** que corre sobre el Host PC, la aplicación que corre sobre el Host adicionalmente se encarga de empaquetarla a través de las capas superiores de protocolo TCP/IP (IP, TCP-UDP) y retransmitirlas a través de la red LAN. El Host PC asume el rol de proxy para el dispositivo Bluetooth que desea acceder a Internet, a su vez cuando, el Host PC accede a Internet, el Proxy actúa como cliente e intercepta la información enviada por algún usuario utilizando en protocolo TCP/IP y redirecciónа esta información sobre en protocolo Bluetooth (ver figura 7).

La figura 8 muestra la implementación final de la tecnología Bluetooth en un dispositivo móvil, que no posee éste tipo de conexión.

Figura 8. Implementación Hardware de nuestro modelo de conexión de la tecnología Bluetooth sobre un Dispositivo móvil

7. Conclusiones y trabajos futuros

La implementación obtenida nos permite acceder a tres servicios de Internet, Smtp, Pop3, Ftp. Dado que la aplicación ***Servidor*** ha integrado completamente el protocolo TCP/IP, mientras que la Aplicación ***Cliente*** solo trabaja con la información necesaria requerida para estos servicios y es transportada directamente a las capas del protocolo Bluetooth, se obtienen dos ventajas en éste modelo implementado, la primera es la eficiencia de **throughput** dado que la transmisión de datos Bluetooth solo usa la información que el usuario desea ver en su dispositivo móvil y no se le agrega **headed** innecesario para la comunicación inalámbrica entre la PDA y el Host PC, puesto que la aplicación ***Servidor*** es la encargada de hacer todo el procesamiento de tramas y control de enlace Bluetooth.

La segunda ventaja es el mejoramiento en procesamiento de tareas en el dispositivo móvil, puesto que la forma como se ha distribuido el manejo de tareas de estas dos aplicaciones, hace que recaiga la mayor cantidad de procesamiento de tramas sobre la aplicación Servidor, junto con la responsabilidad de enviar correctamente los datos que corre sobre un Host PC el cual siempre tendrá una arquitectura hardware más robusta para la ejecución de cualquier aplicación Software con respecto a la arquitectura Hardware de un dispositivo móvil, de esta forma la aplicación cliente solo se encarga esencialmente de obtener la información que el usuario desea visualizar, de esta forma el desempeño de procesamiento que ofrece el dispositivo móvil es adecuado, y robusto para la aplicaciones desarrolladas.

El método de conexión entre el modulo Bluetooth y el Host PC limita sustancialmente la velocidad de transmisión, puesto que la tecnologia Bluetooth 2.0 permite trabajar a una tasa entre 2 – 3 Mbps, mientras que la tasa de transferencia del puerto de comunicación serial del Host PC es generalmente de 921.6 Kbps para el puerto RS232. Para las pruebas realizadas con las aplicaciones Smtp y Pop3 se uso una tasa de transferencia de 9.6Kbps, esta tasa de transferencia es aun mucho menor, sin embargo los resultados obtenidos respecto a la velocidad de desempeño de las aplicaciones fueron adecuados, esto es en parte debido a que éste tipo de aplicaciones no requiere transferencia de grandes cantidades de datos, no obstante al hacer uso de la aplicación FTP para la carga y descarga de archivos, se hizo notorio que la tasa de transferencia resultante era pobre, una alternativa para

mejorar el desempeño consiste en aumentar la tasa de transferencia de puerto serial a 921.6 Kbps, teniendo en cuenta que las aplicaciones desarrolladas para dispositivos móviles no están pensadas para soportar transferencia de datos de grandes cantidades., los resultados que se han obtenido permite pensar que éste modelo de diseño pueda ser tomado como referencia para dotar de servicios de Internet a los dispositivos móviles. Por otro lado el desarrollo de la Aplicación utilizada para el control de los dispositivos Bluetooth permite desarrollar nuevas aplicaciones para la comunicación inalámbrica de forma versátil, haciendo que el control del protocolo Bluetooth sea prácticamente transparente para el programador, dado que la aplicación junto con los módulos Bluetooth ofrecen la totalidad de protocolo Bluetooth hasta el nivel RFCOMM donde se emula comunicación serial punto a punto de datos.

Nosotros hemos considerado la exploración de la tecnología Bluetooth como una solución prometedora para abordar el problema de comunicación inalámbrica para la trasmisión de datos y voz, en contraste con otras tecnologías como IrDA donde aun no se han desarrollado aplicaciones robustas usando línea de vista y 802.11 que posee un consumo de potencia mayor. La principal ventaja de Bluetooth es su servicio de descubrimiento de aplicaciones entre dispositivos, dando mayor robustez a la comunicación entre distintos dispositivos que posean las mismas aplicaciones.

La implementación e integración entre la Red Bluetooth y la Red LAN nos ofrece en primera instancia una herramienta muy poderosa para la transmisión de datos orientada al control de procesos, puesto que unimos la robustez del protocolo TCP/IP con la capacidad de transporte de los módulos Bluetooth, adicionalmente nos permite manipular información desde cualquier ordenador a través de Internet, cabe notar que la rata de transmisión obtenida no es lo suficientemente amplia como para convertir esta tecnología en una opción de acceso a Internet adecuado para una fuerte transferencia de datos, sin embargo es una buena herramienta que se puede implementar en los dispositivos Bluetooth para aplicaciones soportadas por Internet con transferencia de información moderadas.

8. Agradecimientos

Agradecemos a Colciencias, entidad responsable en la investigación en ciencia y tecnología en Colombia, por el apoyo otorgado para la financiación y desarrollo de éste proyecto.

9. Referencias

[1] Bluetooth Special Interest Group, http://www.bluetooth.org/

[2] Bluetooth Special Interest Group, "Bluetooth Core", Specification of the Bluetooth System, Version 2.0, Oct 22, 2004.

[3] Theodoros Salonidis, Pravin Bhagwat, Leandros Tassiulas, and Richard LaMaire, "Distributed Topology Construction of Bluetooth Personal Area Networks". Proc. INFO 2001 IEEE.

[4] Nicolas Rouhana, Eric Horlait, "BWIG: Bluetooth Web Internet Gateway" Seventh International Symposium on Computers and Communications 2002 IEEE.

[5] Juan Torres, Álvaro Bernal, "Implementación de una topología de red de dispositivos Bluetooth capaz de acceder a Internet a través de una Red LAN" Séptimo Encuentro de Investigación sobre tecnologías de Información aplicadas a la solución de problemas 2006.

Modelo Multiagente en Sistemas de Misión Critica Aplicado al Control de Trafico Aéreo Bajo el Concepto de Free Flight

Victor Battista[1], Jorge Ierache[1,2], Paola Britos[3,2], Darío Rodriguez[3,2], Ramón García-Martínez[3,2]

1. Instituto de Sistemas Inteligentes y Enseñanza Experimental de la Robótica. Facultad de Informática, Ciencias de la Comunicación y Técnicas Especiales. Universidad de Morón

2. Laboratorio de Sistemas Inteligentes. Facultad de Ingeniería. Universidad de Buenos Aires

3. Centro de Ingeniería del Software e Ingeniería del Conocimiento. Escuela de Postgrado. Instituto Tecnológico de Buenos Aires

jierache@yahoo.com.ar, jierache@unimoron.edu.ar, victor2206@gmail.com, rgm@itba.edu.ar

Resumen

El crecimiento vertiginoso del tráfico aéreo es una problemática a nivel mundial. Las rutas aéreas, con el paso del tiempo se multiplican al ritmo de los nacientes mercados económicos mundiales.. En éste contexto, el cambio de enfoque en el control de tráfico aéreo surge como la solución más ventajosa dada las desventajas o implicancias de otras propuestas, como el incremento de la capacidad de infraestructura del sistema aeroportuario (aeropuertos y pistas de aterrizaje). En este trabajo se presenta una aproximación de un modelo multiagentes bajo una arquitectura basada en creencias, deseos e intenciones de los agentes que implementan el concepto de Free Flight, los aviones, así, son sus propios controladores. Pueden elegir sus propias rutas de navegación, velocidad, altura, regimenes de ascenso y descenso, maniobras. Para el apoyo, análisis y gestión de una ambiente de vuelo libre, se considera la caracterización de un modelo de simulación basado en un entorno de vida artificial a fin de observar la evolución del espacio aéreo bajo un concepto de Free Flight. para una población inicial de aeronaves con intenciones diversas respecto a sus aeropuertos de destino.

1. Introducción

Las tendencias demográficas mundiales, el crecimiento de las ciudades, el desarrollo de nuevos centros urbanos, la evolución de la tecnología en la aviación, han llevado al transporte aéreo a una posición privilegiada de crecimiento continuo.

En el campo demográfico, 14 centros urbanos del mundo tendrán más de 15 millones de habitantes; cuatro de ellos, más de 20 millones (Tokio, con alrededor de 30 millones, seguido por Lagos, Bombay y Sao Paulo) [19]. En el pasado, como en el presente, Europa, Asia Oriental y Estados Unidos son referentes económicos y generadores de tráfico aéreo. Sin embargo, en los últimos años han ganado mucho terreno la India, China y el sur de Sudamérica. Los dos primeros apoyados por una fuerte expansión económica y la última por el potencial turístico que ostenta.

La aviación comercial desde sus comienzos al día de hoy ha avanzado notablemente logrando un medio de transporte seguro y efectivo. Hacia finales de febrero de 2006, la empresa Boeing anunció que había entregado 5.009 aviones de la familia '737' [5]

En la actualidad, el crecimiento del tráfico aéreo es analizado intensamente. El congestionamiento ha alcanzado límites tales que alrededor del mundo distintos pronósticos señalan el colapso de los sistemas de control de vuelo en los años próximos. Las rutas aéreas con el paso del tiempo se multiplican, el tránsito aéreo es cada vez más intenso y las pérdidas económicas a causa de los retrasos y congestionamiento son millonarias. [6]

La solución clásica a este tipo de problemáticas era el incremento del número de aeropuertos y la creación de nuevas pistas de aterrizajes.

Con esta perspectiva de la realidad, un cambio de enfoque en el control de tráfico aéreo surge como la solución más ventajosa dada las desventajas o implicancias de las propuestas clásicas.

En el presente trabajo se describe aproximación a un modelo al control del tráfico aéreo diferente al enfoque actual en uso. La propuesta es un modelo multiagente considerando los roles mas significativos e incorporando el concepto de vuelo libre [10], [17] para la elección de las rutas de vuelo.

2. Control aéreo inteligente

El modelo de sistema multiagente que se desea desarrollar se desenvuelve en un ambiente altamente ágil y dinámico. Si bien, los componentes identificados son unos pocos, el modelo puede ser especialmente complejo de acuerdo a la cantidad de instancia de un elemento esencial: las aeronaves. Cada una de ellas, en el sistema, se modelizan a través de un agente "Piloto Aeronave". El mismo, por definición deberá mostrar un comportamiento flexible. Dicha flexibilidad le permitirá, a pesar de las distintas evoluciones que presente el ambiente, alcanzar su objetivo: lograr el aterrizaje a tiempo en su destino.

Si pensamos en los distintos desafíos que un agente "Piloto Aeronave" puede encontrar durante un vuelo veremos que entran a jugar un amplio rango de eventos que pueden influenciar de forma positiva o negativa. Por ejemplo, supongamos que una aeronave parte de un aeropuerto A y se dirige a un aeropuerto B. Al principio, el agente determinará una ruta de vuelo posible según un estudio primario de sus creencias en relación al ambiente: condiciones meteorológicas, congestionamiento del aeropuerto, distancias de vuelo, rutas de vuelo tradicionales, su capacidad de combustible, entre otras. Así el agente aeronave parte con una intención/plan de vuelo para cumplir con su deseo de arribar al aeropuerto de destino en el menor tiempo con el menor consumo de combustible. Sin embargo, el sistema puede variar drásticamente, en ocasiones espontáneamente apelando a las características reactivas del Agente; otras veces, paulatinamente dando lugar a un comportamiento más pro-activo. La formación de tormentas eléctricas, la aparición de aeronaves, las condiciones meteorológicas en general, son sólo algunas de las distintas manifestaciones del ambiente en el cual se moviliza el agente "Piloto Aeronave".

Finalmente, la aeronave se aproxima al espacio aéreo correspondiente al área terminal. Es en este momento, donde quizás la capacidad de negociación del agente "Piloto Aeronave" cobra vital importancia. La aeronave que se traslada de un aeropuerto A a otro B debe aterrizar. El primer problema es que la pista de aterrizaje es de por si un recurso finito y su disponibilidad sólo ocurre en pequeñas "ventanas" de tiempo. La pista de aterrizaje es un recurso que todas las aeronaves deberán compartir en un instante de tiempo particular, sólo una de ellas podrá hacer uso de la misma para despegar o aterrizar. Esto genera una situación de competencia entre los distintos agente "Piloto Aeronave".

Puede resultar que la aeronave a su llegada al aeropuerto se encuentre sólo esperando autorización para aterrizar, de modo que no tendrá problemas en hacerlo. Sin embargo, sabemos que esto no será así en las mayorías de los casos; precisamente es esta la motivación que da lugar a este trabajo de investigación.

Frecuentemente, el Agente aeronave deberá llegar a un acuerdo tripartito con otros agentes aeronaves y controladores aéreos, teniendo en cuenta la disponibilidad de la pista y rutas de vuelo, el combustible con el que cuenta cada aeronave, su autonomía, afluencia de trafico, entre otros aspectos para conseguir un lugar en la cola de aterrizajes.

El objetivo del sistema se centra en alcanzar la maximización del aprovechamiento de los recursos aeroportuarios en orden de aliviar la congestión en los mismos. Además, el sistema deberá gestionar los recursos para que cada uno de los aviones puedan realizar sus vuelos de forma segura.

Ante el crecimiento de la congestión del tráfico aéreo la autoridad de aviación civil australiana[1] ha dado origen al programa de mejoramiento del manejo de tráfico aéreo. En un desarrollo conjunto, con el Instituto de Inteligencia Artificial Australiano, dio lugar a OASIS[2][15].

Un sistema de administración de tráfico aéreo prototipo desarrollado para el aeropuerto Kingsford Smith de Sydney. Dicho proyecto fue tomado como referencia para este trabajo.

Considerando el nuevo enfoque innovador sobre la problemática, se ha considerado el concepto importante y de especial auge dentro del área del control de vuelo: el vuelo libre (Free Flight).

El control de vuelo clásico, utilizado, se ha dejado de lado en favor de la implementación del vuelo libre. Los aviones, así, son sus propios controladores. Pueden elegir sus propias rutas de navegación, velocidad, altura, regimenes de ascenso y descenso, maniobras, en este contexto el objetivo del sistema es la planificación de los aterrizajes de las aeronaves como se detalla en la figura 1.

El sistema le brinda total libertad al piloto para que vuele según sus convicciones, sin embargo, un sistema de control de vuelo descentralizado analiza el ambiente

[1] Denominada *Airservices Australia*
[2] *Optimal Aircraft Sequencing using Intelligent Scheduling*

con el objetivo de velar por el cumplimiento del objetivo final: un vuelo seguro y sin inconvenientes. En consecuencia se pretende alcanzar un alto grado de cooperación y la capacidad de trabajar por un mismo objetivo.

Figura 1. Objetivos del sistema

3. Free flight

Free Flight [2], propone un sistema de control que abandona el monitoreo centralizado, desde tierra, del tráfico para adoptar una arquitectura descentralizada y altamente cooperativa. El concepto descansa sobre dispositivos digitales altamente precisos para la ubicación de aeronaves (GPS: Global Positioning System), un sistema de comunicación de datos digital, que brinda información a todos los involucrados, apoyados con la automatización de gran parte de las tareas del los controladores aéreos.

Bajo el "vuelo libre" los controladores no acaparan responsabilidades sino que ellas son delegadas a los pilotos. Los pilotos son sus propios controladores de vuelo, son libres de escoger trayectorias de vuelos, altitudes, velocidades como alguna vez lo hicieron antes de que existieran los primeros sistemas de control. El "vuelo libre" podría reducir la congestión de los aeropuertos más concurridos elevando el promedio de aterrizajes por unidad de tiempo.

El concepto de free Flight [7] se define como una capacidad de operación de vuelo bajo condiciones de vuelo por instrumentos en el cual los operadores tienen la libertad de elegir su trayectoria y velocidad en tiempo real. Las restricciones al tráfico aéreo se imponen sólo para asegurar la separación, para evitar el exceso de capacidad de aeropuerto, para prevenir vuelo no autorizado a través espacio aéreo de uso restringido/prohibido, y para garantizar la seguridad del vuelo.

4. Modelo multiagente para control aéreo

El modelo propuesto se compone principalmente por cinco roles: Vigilancia y Control, Planificador, Meteorólogo, Controlador, Piloto Aeronave; según se detalla en la figura 2.

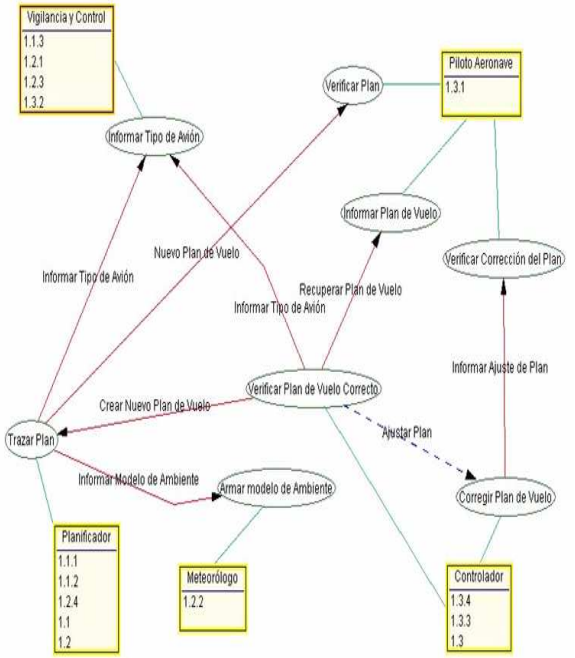


Figura 2. Modelo de roles del sistema

El rol "Vigilancia y Control" es el encargado de individualizar cada aeronave que ingresa en el espacio aéreo de su área de control. Deberá, entre otras cosas, informar la posición actual de la aeronave e identificarla.

El rol "Planificador" sugiere un plan de vuelo eficaz con base en el análisis de la aeronave, las condiciones climáticas y la programación de aterrizaje. Se encarga de armar un plan de vuelo eficaz para la aeronave y lo informa como directivas similares al formato de una 'lista de tareas' pendientes.

El rol "Meteorólogo" actualiza la información ambiental a través de los datos que puedan recoger los distintos aviones en el aire junto con la que él mismo posee y además brindan los sensores meteorológicos del área.

El rol "Controlador" es el designado para hacer el seguimiento del desempeño de los aviones próximos al aeropuerto. Además, es el encargado de detectar cualquier tipo de inconveniente, accidente o problema que pudiera surgir. Deberá corregir los planes de vuelo evaluando la situación actual de la aeronave y, a partir de allí, componer una serie de modificaciones al plan de vuelo original. Periódicamente verificará que el la trayectoria del aeronave se encuentre dentro del

margen de error aceptable para el plan de vuelo estimado.

El rol “Piloto Aeronave” se encargará de tomar las decisiones de vuelo, por ejemplo, sobre: la velocidad, la altitud, la dirección. Verificará que el plan de vuelo estimado esté dentro de las posibilidades técnicas de la aeronave. Mantendrá actualizada, y disponible para el sistema, toda la información del plan de vuelo de la aeronave. Todas las correcciones al plan de vuelo original serán verificadas con el objetivo de constatar que sean compatibles con las capacidades de la aeronave.

El comportamiento de un agente aislado queda determinado por sus motivaciones individuales, sus creencias acerca del mundo y sus propias habilidades. De este modo, esta caracterización resulta insuficiente para modelar a un agente con una actitud cooperativa. La coordinación de los agentes, al igual que en otros dominios, es un aspecto importante para lograr un comportamiento coherente. La distribución de los datos y el control entre varios agentes autónomos a menudo deriva en situaciones de conflictos como consecuencia de la existencia de distintos puntos de interés, los recursos escasos y las motivaciones particulares u objetivos.

El aspecto de la comunicación entre los agentes es un factor fundamental a considerar. La tendencia actual, es utilizar lenguajes de comunicación de agentes basados en la teoría de “speech acts” como por ejemplo KQML [14] y FIPA ACL [8]. En el caso de JADE, el Framework de desarrollo de agentes provee un amplio soporte para las comunicaciones entre agentes a partir del estándar FIPA ACL.

Para ejemplificar este esquema, consideremos el caso en que dos aviones se aproximen en una área de control a un aeropuerto y se comunican con el controlador del área para establecer su lugar en la cola de aterrizajes. En este caso, ambos agentes “Piloto Aeronave” ingresan en una situación de competencia en relación a un recurso escaso, como lo es la pista de aterrizaje. Si bien éste no es un recurso agotable tiene la particularidad que puede estar disponible sólo para un agente en un instante de tiempo puntual. Cada uno de los agentes “Piloto Aeronave” tiene como uno de sus objetivos principales el ahorro de combustible y la construcción de un plan de vuelo que apunte a la maximización de este recurso. Por esta condición, al momento de ingresar al área de control, el agente “Piloto Aeronave” deberá interactuar con el agente “Vigilancia y Control” y para conseguir un lugar en la cola de aterrizajes deberá negociar con el “Planificador” quien tiene como objetivo lograr la mejor configuración del espacio aéreo para todos los agentes involucrados, según se detalla en la figura 3.

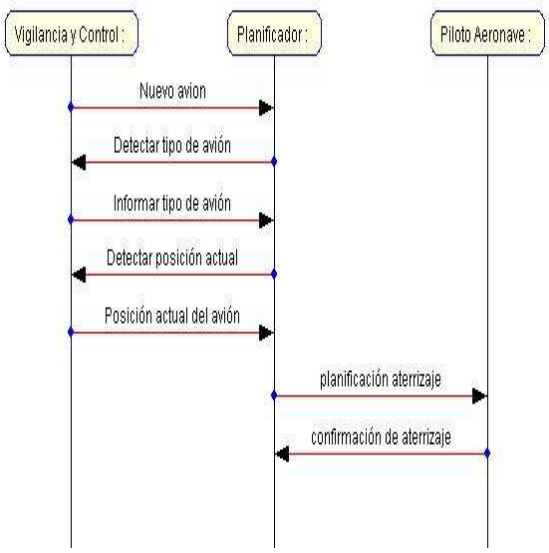


Figura 3. Diagrama de secuencias correspondiente al ingreso de un aeronave

Cada uno de los agentes que componen el sistema han sido modelados desde la perspectiva BDI [4].

El agente “Vigilancia y Control” tiene el deseo, objetivo de lograr individualizar e identificar cada aeronave que se desplaza en el espacio aéreo. Sus creencias se conforman de la información que le suministran distintos sensores de vigilancia y control del espacio aéreo. Finalmente, sus intenciones o acciones se focalizan en la transmisión de información actualizada acerca de los aviones en su espacio aéreo.

El agente “Planificador” posee el deseo de programar los aterrizajes para lograr que el agente “Piloto Aeronave” pueda trazar un plan de vuelo eficaz. Sus creencias se basan en la meteorología, sus conocimientos acerca de la aeronave en cuestión, su autonomía de vuelo y la disponibilidad del recurso de la pista de aterrizaje, entre otras. Sus intenciones son la gestión de la cola de aterrizajes en el aeropuerto y sugerir al piloto las maniobras más convenientes para su aproximación dentro del área terminal con destino a su aeropuerto de aterrizaje.

El agente “Meteorólogo” tiene el objetivo de actualizar la información ambiental a través de la información que puedan recoger los distintos aviones en el aire (sus creencias), apoyado con los sistemas meteorológicos, su intención es brindar un modelo de ambiente meteorológico.

El agente “Controlador” debe detectar los inconveniente, potenciales incidente o problema que pudiera surgir con las rutas de vuelo de los agentes “Pilotos aeronaves”. Recabará información a través de

mensajes con los distintos agentes “Pilotos aeronaves” en vuelo y con el agente “Vigilancia y Control” quien le brindara la situación del espacio aéreo. Entre sus actividades se enumeran la evaluación de las rutas de vuelo y la configuración del espacio aéreo y la alerta de colisiones al agente “Piloto aeronave”.

El agente “Piloto Aeronave” tiene el deseo de alcanzar un aeropuerto de destino llevando adelante un vuelo seguro y eficaz sin complicaciones. Se nutre de la información que obtiene del amplio rango de instrumental de vuelo (TCAS [3], GPS) para generar sus creencias. Para alcanzar su objetivo llevará adelante verificaciones periódicas del plan de vuelo y la corrección del mismo ante una alerta de colisión.

5. Modelo de simulación para el control aéreo en ambiente de vuelo libre

En este contexto según se representa sintéticamente en la figura 4, se propone una aproximación para el ambiente de vuelo libre caracterizado en un entorno de vida artificial [1], seleccionado este por su capacidad para la simulación de sistemas descentralizados donde los autómatas celulares resultan un marco para explorar fenómenos de autoorganización, presentes en el concepto de vuelo libre, donde no se cuenta con diseñador centralizado para el control aéreo. Observando las características de un hábitat de vida artificial se considera como Depredadores a los agentes “Piloto aeronave” y como Presas se presentan dos tipos constituidos por la presa principal denominada como presa tipo pista de aterrizaje, correspondiente a su destino y la presa tipo espacio aéreo (X,Y,Z) a ocupar por un agente “Piloto aeronave” durante un tiempo (T), suficiente para asegurar las separaciones entre aeronaves y evitar colisiones, si bien son un recurso renovable como alimentos para sus depredadores, se encuentran agotados durante un margen de tiempo al ser consumidos por otro depredador representado por el agente “Piloto aeronave”.

Como Hábitat de los Depredadores y Presas se considera el espacio aéreo y aeroportuario a través de parcelas o celdas organizadas en una grilla de modo similar al de un autómata celular con depredadores que deambulan por encima (aeronaves que vuelan en el espacio aéreo), cada parcela se caracteriza por la acción de los agentes Planificador, Controlador, Meteorólogo, Vigilancia y Control y la presencia de una Presa representada por el espacio aéreo que se corresponden con un juego de valores limites XYZ de la celda salvo los casos de espacios prohibidos de vuelo o restringidos temporalmente para el uso en los cuales no se consideran presas, las celdas podrán estar ocupadas por depredadores agentes “Piloto aeronave” respetando los liftes de separación entre aeronaves considerados para garantizar la seguridad aérea, y también por presas del tipo pista de aterrizaje.

La energía de los depredadores (agentes “Pilotos aeronaves”) se considera en función del consumo de combustible, pudiendo agotarla o no para poder cumplir su vuelo en los márgenes de tiempo establecidos, ocurriendo la muerte del depredador si se agota su combustible o aterriza fuera de los márgenes de tiempo establecidos.

El observador mira a lo depredadores (agentes “Pilotos aeronaves”) y administrar las tazas de creación de nuevos depredadores y su agotamiento en función de los márgenes de tiempo que establece para el consumo de su presa, (representado por el arribo de la aeronave a su pista del aeropuerto de destino), además el observador monitorea la actividad de las presas y de las parcelas o celdas del hábitat (espacio aéreo).

La extinción de la especie se daría por el agotamiento de la energía de los depredadores, y el consumo de sus presas en un instante de tiempo de acuerdo a los márgenes seleccionados por el observador para lo cual los depredadores agentes “Pilotos aeronaves” no puedan consumir a su presa elegida (espacio aéreo X,Y,Z durante un tiempo T, pista de aterrizaje del aeropuerto de destino).

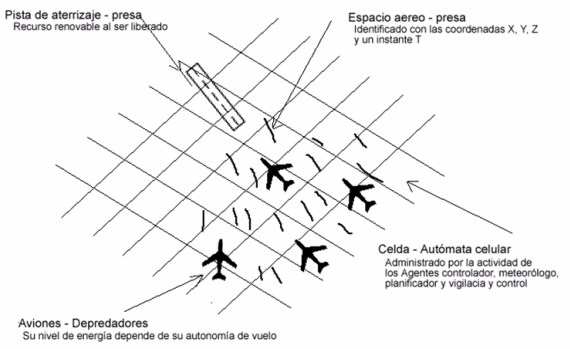


Figura 4. Ambiente de vuelo libre modelado en un entorno de vida artificial.

Para el desarrollo del modelo se analizaron herramientas orientadas a agentes como JADE [12], la especificación FIPA ACL [8] para mensajes interagentes, Protegè [18] (para crear una ontología del sistema), MaSE [16] como metodología., StarLogo [20] para vida artificial. En [13] se presenta los resultados de una simulación basada en teoría de juegos para la toma de decisiones en vuelo libre por parte de las aeronaves considerando una actitud altruista especifica para la negociación en un ambiente

multiagente a fin de evitar incidentes aéreos entre aeronaves en un entorno de vuelo libre, sin embargo en nuestra propuesta se considera que las aeronaves cuentan con sistemas TCAS (sistema de alerta de colisión entre aeronaves) para alertar incidentes aéreos en su espacio aéreo y realizar las maniobras de evasión a fin de mantener una separación vertical y horizontal considerada para las aeronaves que conforman el hábitat de simulación en un entorno de vida artificial.

La simulación implementada del ambiente de vida artificial se caracteriza por facilitar:

- Cantidad de aviones. Máximo diez (10).
- Cantidad de aeropuertos. La simulación se puede llevar adelante con dos (2) o tres (3) aeropuertos.
- Energía Máxima de los aviones. Este concepto es análogo a la cantidad de combustible que el avión posee para realizar su vuelo. Máximo parametrizable: cuatrocientos (400)
- Consumo de energía. Es la cantidad de energía que un avión consume al viajar una (1) unidad de distancia en busca de alcanzar su destino.
- Capacidad del aeropuerto. Es la cantidad de aviones que podrán permanecer en un aeropuerto, en un instante de tiempo dado, antes de volver a despegar. Máximo diez (10).

El hábitat simulado evoluciono enmarcado en las siguientes reglas:

- Los aviones tienen como objetivo lograr su aterrizaje en un destino conocido.
- Los aviones tienen un consumo de energía ligado a la distancia recorrida en busca de un aeropuerto. Al acabarse dicha energía si el avión no alcanza su destino principal o alternativa, sufre el siniestro.
- Los aviones no vuelan a una altura predefinida. Cada avión elige su altitud arbitrariamente.
- Cuando un avión llega a destino sólo hará el aterrizaje si la capacidad del aeropuerto no está colmada. De ser así, el avión recibe un nuevo destino de alternativa.
- Cuando el avión logra el aterrizaje permanecerá en "reposo" en el aeropuerto durante un tiempo, equivalente al que estuvo en vuelo, con el objetivo de recuperar la energía perdida. Mientras el avión permanece en el aeropuerto ocupa una plaza de capacidad disponible del aeropuerto.
- Los aviones en vuelo deben respetar un límite de separación, tanto vertical como horizontal. Esto define un área de seguridad que permite que el avión vuele sin riesgos de colisión. Esto es, un avión sólo podrá volar en una dirección si al hacerlo no ingresase en dicha área correspondiente a otro avión volando en sus cercanías

Se presenta en la figura 5 la pantalla principal del simulador, en la que se genera el ambiente para iniciar la simulación utilizando los valores parametrizables detallados.

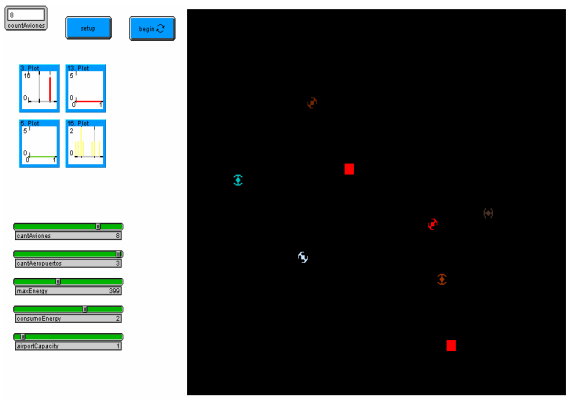


Figura 5. Pantalla de simulación en Starlogo.

La interfase de representación de la simulación cuenta, además con un monitor ubicado en el sector superior para medir el comportamiento de las aeronaves en su hábitat en tiempo real, brindando información en primer lugar: relacionada con la disponibilidad de combustible/energía de los aviones, según se detalla en la figura 6

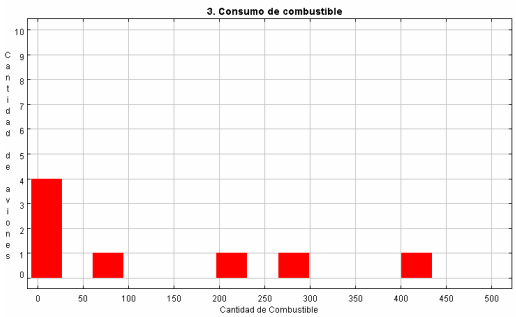


Figura 6. Disponibilidad de combustible.

El segundo se brinda información de la evolución de la separación vertical promedio calculada a partir de la separación con respecto al avión más próximo que registra cada avión, según se detalla en la figura 7

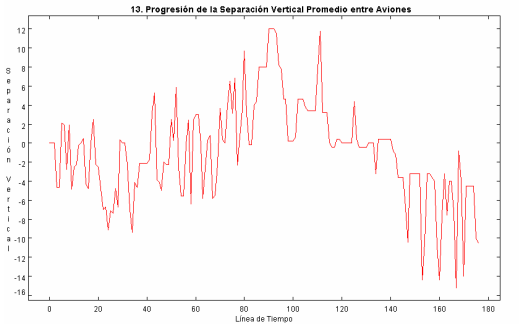


Figura 7. Evolución de la separación vertical

El tercer lugar se brinda información de la evolución de la separación horizontal promedio calculada a partir de la separación con respecto al avión más próximo que registra cada avión. Según se detalla en la figura 8.

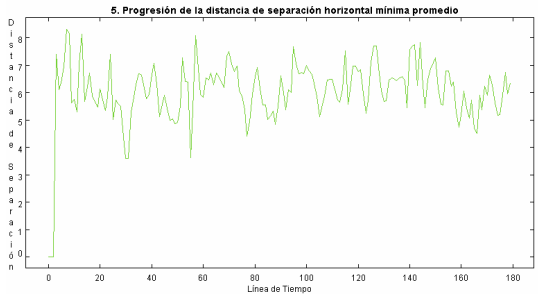


Figura 8. Evolución de la separación horizontal

El cuarto lugar se brinda información de la distribución de la variable altitud de los diferentes aviones en vuelo. Según se detalla en la figura 9

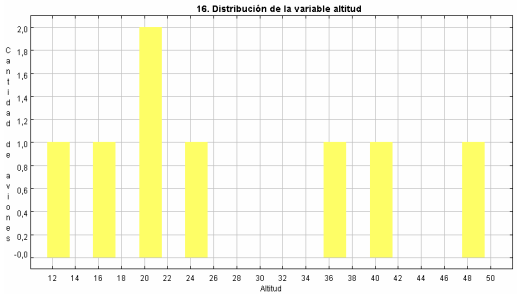


Figura 9. Distribución de la altitud entre aviones.

7. JADE (Java Agent Development Framework)

JADE es un middleware que proporciona un entorno para desarrollar agentes, que explota el potencial de la plataforma java. Otro aspecto importante es que el Framework, ofrece como servicios propios funcionalidades básicas que requiere un agente y permite centrarse en el desarrollo de los aspectos lógico de los agentes

Algunas de las características que presenta JADE son las siguientes:

- P2P: Arquitectura peer-to-peer, cada agente puede tomar la iniciativa en una comunicación o bien responder a peticiones que le hagan otros agentes.
- Portabilidad: La API que proporciona JADE es independientemente de la red sobre la que va a operar, así como de la versión de Java utilizada, teniendo la misma API para J2EE [9] , J2SE [10] y J2ME [11] .
- JADE también brinda soporte para agentes móviles. Un agente JADE puede interrumpir en un momento dado su ejecución, migrar a otro host (sin necesidad de que el código esté previamente en el host destino) y continuar la ejecución en el mismo punto en el que la interrumpió. Esto permite un balanceo de carga ya que permite a los agentes migrar hacia hosts menos cargados.
- El entorno de ejecución proporciona un marco donde poder ejecutar los agentes y herramientas gráficas para su monitorización y depuración.

Finalmente, JADE simplifica la comunicación y la cooperación entre los agentes; los agentes JADE pueden controlar su propio ciclo de vida, y pueden ser programados para que dejen de funcionar o empiecen a hacerlo dependiendo del estado del sistema y de la función que debe realizar el agente; JADE cumple con la especificación de FIPA [8], por lo tanto, puede comunicarse con agentes realizados en otros entornos pero que utilicen este estándar.

8. Conclusión y futuras líneas de investigación

La capacidad de coordinación y de cooperación de los agentes para cumplir con los objetivos facilita enormemente la resolución de los problemas propuestos. En este trabajo se presento una aproximación de un modelo multiagentes bajo una arquitectura basada en creencias, deseos e intenciones de los agentes que implementan el concepto de Free Flight, los aviones, así, son sus propios controladores. Pueden elegir sus propias rutas de navegación, velocidad, altura, regimenes de ascenso y descenso, maniobras. Para el apoyo, análisis y gestión de una ambiente de vuelo libre dada sus características propias de un sistema descentralizado sin un director central a cargo del control aéreo, se considero la caracterización de un modelo de simulación basado en un entorno de vida artificial a fin de observar la evolución del

espacio aéreo bajo un concepto de Free Flight, para una población inicial de aeronaves con intenciones diversas, respecto a sus aeropuertos de destino.

Dentro de las futuras líneas de investigación se considera la elaboración de un Framework para correr el modelo multiagente destinado al apoyo, análisis y gestión de una ambiente de vuelo libre, la elaboración de ontologías para el concepto de Free Flight

8. Referencias

[1] Adami, C, *Introduction to Artificial Life*, Springer-Verlag, Nueva Cork,1998

[2] Air Traffic Control (ATC). *Aspects of Free Flight*. http://www.freeflightatm.org/, vigente al 01.06.06

[3] ALLSTAR. *Aeronautics Learning Laboratory for Science Technology and Research*. http://www.allstar.fiu.edu/AERO/TCAS.htm, vigente al 01.06.06

[4] Anand S. Rao, Michael P. Georgeoff. *BDI Agents: From Theory to Practice*. http://www.agent.ai/doc/upload/200302/rao95.pdf, vigente al 01.06.06

[5] Boeing Company. http://www.boeing.com/, vigente al 01.06.06

[6] Federal Aviation Administration. http://www.faa.gov/, vigente al 01.06.06

[7] Free Flight with airbone separation assurance. http://www2.nlr.nl/public/hosted-sites/freeflight/, vigente al 01.06.06

[8] FIPA. *Foundation for Intelligent Physical Agents*. http://www.fipa.org/ vigente al 01.06.06

[9] J2EE. *Java 2 Platform, Enterprise Edition*. http://java.sun.com/javaee/, vigente al 01.06.06

[10] J2SE. *Java 2 Platform, Standard Edition*. http://java.sun.com/javase/, visitado en junio de 2006

[11] J2ME. *Java 2 Platform, Micro Edition*. http://java.sun.com/javame/, visitado vigente al 01.06.06

[12] JADE. *Java Agent DEvelopment Framework*. http://jade.tilab.com/, vigente al 01.06.06

[13] Hill, J., Johnson, R., Archibald, J., Frost, R., Stirling, W. *A cooperative multi-agent approach to free flight*, Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems, pp. 1083 - 1090. 2005

[14] KQML. *Knowledge Query and Manipulation Language*, http://www.cs.umbc.edu/kqml vigente al 01.06.06Ljungberg, M, Lucas, A. 1992.

[15] Ljungberg, M., Lucas, A. 1992. *The OASIS Air Traffic Management System*. Proceedings of the Second Pacific Rim International Conference on Artificial Intelligence.

[16] MaSE.. *Multiagent Systems Engineering*. http://macr.cis.ksu.edu/projects/mase.htm, vigente al 01.06.06

[17] NASA. *Air Traffic Controller Performance in Free Flight Simulation*. http://human-factors.arc.nasa.gov/cognition/research/freeflgt.html, vigente al 01.06.06

[18] Protege. *Proyect Protegè*. http://protege.stanford.edu/, vigente al 01.06.06

[19] Davies, R. *Orientaciones del Transporte Aéreo en el Siglo XXI: Las Lecciones de la Historia*. http://usinfo.state.gov/ journals/ites/1000/ijes/trans6.htm, vigente al 01.06.06

[20] Starlogo. *StarLogo on the Web*. http://education.mit.edu/starlogo/, vigente al 01.06.06

El Problema Cinemático en Manipuladores Robóticos Industriales
Un Abordaje de Solución mediante Redes Neuronales Artificiales

Alejandro Hossian, Enrique Sierra, Enrique Fernández, Paola Britos, Ramón García Martínez
Departamento Electrotecnia. Facultad de Ingeniería. Universidad Nacional del Comahue
Centro de Ingeniería del Software e Ingeniería del Conocimiento. Escuela de Postgrado. ITBA.
Laboratorio de Sistemas Inteligentes. Facultad de Ingeniería. Universidad de Buenos Aires.
hossi@ciudad.com.ar, esierra@uncoma.edu.a, {enfernan,pbritos,rgm}@itba.edu.ar

Resumen

La robótica es un campo relativamente joven de la tecnología moderna que atraviesa las fronteras de la ingeniería tradicional. El aporte fundamental de este artículo consiste en ofrecer una solución al problema cinemático inverso existente en manipuladores robóticos mediante el uso de redes neuronales artificiales. La solución encontrada ofrece significativas ventajas en comparación con los métodos usualmente empleados.

1. Introducción

En el campo de la robótica, uno de los manipuladores más utilizados es el angular o antropomórfico, el cual posee todas sus articulaciones de rotación o esféricas, tal como se ilustra en la Figura 1. Si además en el extremo del manipulador se coloca una articulación esférica, se obtienen seis grados de libertad, los cuales permiten la colocación del extremo del manipulador en un amplio espacio de trabajo. De este modo, el manipulador propuesto posee seis grados de libertad.

Los tres primeros están referidos a articulaciones de rotación y los últimos tres grados se refieren a los de la articulación esférica del extremo. En la Figura 2 se muestra una representación del manipulador considerado como la unión de eslabones (uniones negras) y articulaciones (cilindros en azul). De este modo y en base a su geometría y dimensiones, el espacio de trabajo accesible por el manipulador (workspace) puede representarse mediante una esfera con un radio máximo de 0.8636 m, la cual es atravesada en forma vertical por un cilindro de 0.15 m de radio, que corresponde a la zona interna del espacio de trabajo que el manipulador no puede alcanzar. La Figura 3 muestra la forma que adopta este espacio de trabajo.

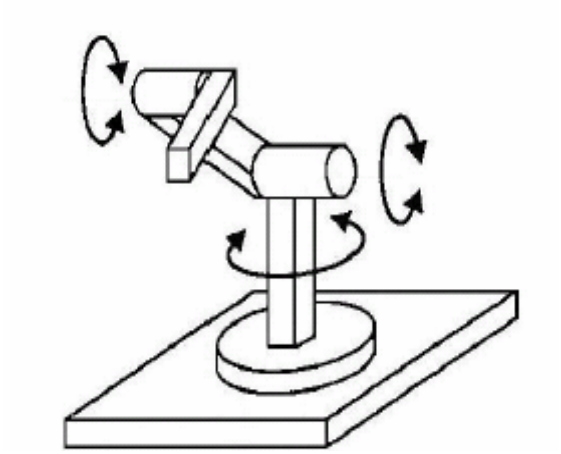


Figura 1. Manipulador Robótico Antropomórfico

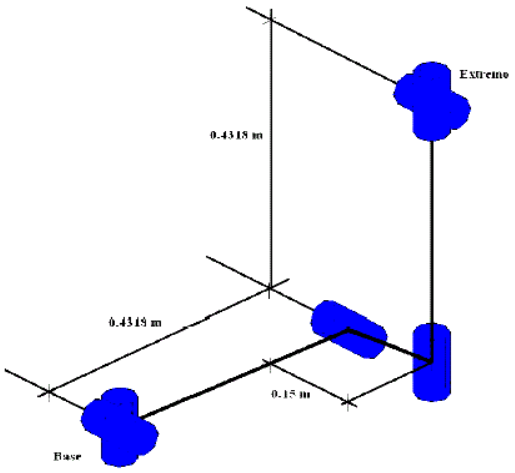


Figura 2. Estructura y dimensiones del manipulador

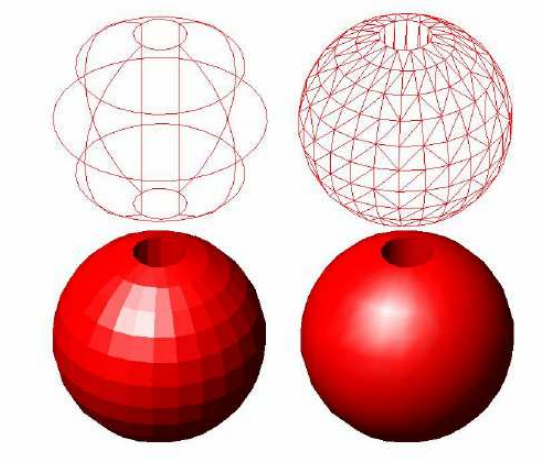


Figura 3. Espacio de trabajo accesible para el manipulador

2. Resolución del problema cinemático directo

La cinemática de posición directa consiste en establecer la posición del extremo del robot en función de los valores de los parámetros de las articulaciones [1]. A efectos de resolverlo, es necesario llevar a cabo los 16 pasos del algoritmo de Denavid Hartenberg [2] para obtener el modelo cinemático directo. Conforme a este algoritmo, lo primero que debe realizarse es la enumeración de los eslabones y las articulaciones.

Todas las articulaciones con sus respectivos sistemas de ejes asociados se pueden observar en la Figura 4. En esta ilustración, el eje z corresponde al propio eje de giro de la articulación.

Una vez obtenidos los parámetros de cada articulación, se pueden calcular los parámetros de Denavid-Hartenberg relativos al tamaño, forma y posición de los eslabones. Estos parámetros se definen como:

a_i: distancia entre los ejes i e i+1 de las articulaciones a lo largo de la normal común. Este parámetro define en cierto modo el tamaño del eslabón, por lo que se lo conoce como "longitud del eslabón".

α_i: ángulo que existiría entre los ejes i e i+1 de las articulaciones si éstos se cortasen los puntos de corte de la línea normal común. Este parámetro en cierto modo mide la forma del eslabón a través del ángulo que sobre el mismo se encuentra girado, por lo que se lo conoce como "ángulo de torsión del eslabón".

d_i: distancia entre las intersecciones de las normales comunes al eje de la articulación i, medida a lo largo de dicho eje. Esta medida en cierto modo expresa la distancia entre los dos eslabones, marcada por el tamaño y forma de la articulación, por lo que se denomina "longitud articular".

θ_i: ángulo que existiría entre las líneas normales comunes al eje de la articulación i si se cortasen en el mismo punto del eje de la articulación. De alguna forma expresa el ángulo que forman los dos eslabones, marcado nuevamente por la forma de la articulación, por lo que se denomina "ángulo articular".

De esta forma es posible elaborar para el robot considerado una tabla con los valores de dichos parámetros. Esta tabla se ilustra en la Tabla I.

Parámetros	θ	a	d	α
Articulación N° 1	θ_1	0	0	90°
Articulación N° 2	θ_2	0.4318 m	0	0
Articulación N° 3	θ_3	0.02 m	0.15 m	-90°
Articulación N° 4	θ_4	0	0.4318 m	90°
Articulación N° 5	θ_5	0	0	-90
Articulación N° 6	θ_6	0	0	0

Tabla I. Valores de los parámetros de Denavid-Hartenberg

Para representar una traslación y/o una rotación, estos parámetros se utilizan en la construcción de la matriz de transformación homogénea:

$$ {}^{i-1}T_i = \begin{bmatrix} \cos\theta_i & -\cos\alpha_i . sen\theta_i & sen\alpha_i sen\theta_i & d_i . \cos\theta_i \\ sen\theta_i & \cos\alpha_i . \cos\theta_i & -sen\alpha_i \cos\theta_i & d_i sen\theta_i \\ 0 & sen\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

De esta forma se obtiene una representación de los sistemas de referencia de cada articulación (incluyendo el extremo del manipulador) con respecto a las anteriores (incluyendo la base del manipulador). El ángulo θ de cada articulación corresponde al ángulo de rotación alrededor del eje z, es por ello que en la Figura 4 se comprueba que los sistemas de ejes son tales que el eje z es coincidente con el eje de revolución de la articulación. Las matrices de transformación homogénea [3] pueden construirse para cada una de las articulaciones.

Con estas seis matrices se puede obtener la orientación y la posición de cada una de las articulaciones con respecto a la anterior o,

resolviendo el producto de ellas, la posición y orientación de cada articulación con respecto a la base del manipulador. Este tipo de manipuladores posee lo que se denomina desacoplo cinemático. Esto es, existen tres articulaciones que son las encargadas del posicionamiento del extremo del manipulador. De esta manera, si solo interesa la posición espacial del extremo del manipulador, basta con conocer la matriz de transformación homogénea que relaciona la base del manipulador con la tercera articulación.

Se ha diseñado un software, denominado *Armbot*, que corre bajo el ambiente MATLAB 6.0 que contiene la resolución de los aspectos cinemáticos del manipulador. Dentro de este ambiente de simulación se encuentra la resolución del problema cinemático directo. La pantalla de interface que resuelve este problema en el software diseñado tiene la forma que se observa en la Figura 5.

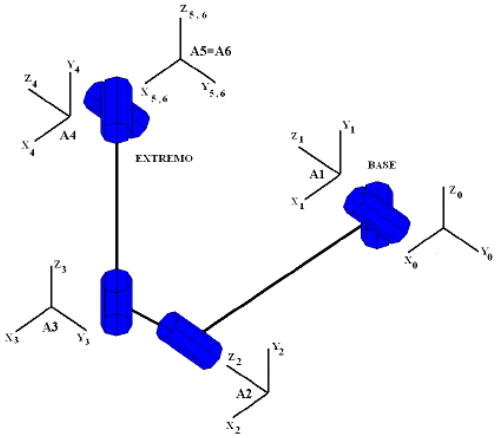


Figura 4. Enumeración de las articulaciones

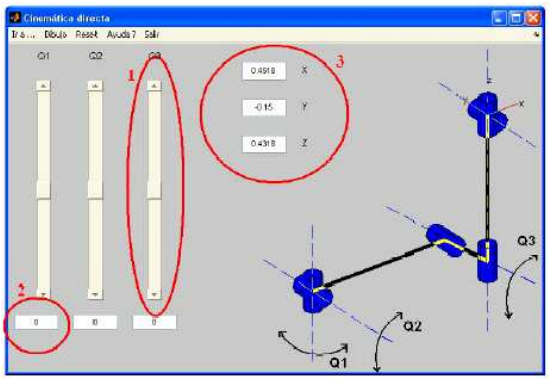


Figura 5. Interfase para la resolución del problema cinemático directo

En el software, mediante el deslizamiento de las barras deslizadoras o sliders (marcadas como 1) se logra variar los ángulos de las articulaciones que se ilustran en el gráfico. Otra manera de modificar los ángulos de las articulaciones es mediante las cajas de texto o box-text (marcados como 2). Estos ángulos están expresados en radianes. Por cada variación de estos ángulos, se calcula la posición del extremo del manipulador con respecto a la base del mismo. Estos datos están expresados en las cajas de texto XYZ (marcadas como 3). En la Figura 6 se ilustra la secuencia de pasos que utiliza el programa para calcular la cinemática directa del manipulador. Posteriormente, es posible observar la posición del manipulador cuando se varían los ángulos de las tres primeras articulaciones, tal como se ilustra en la Figura 7.

3. Resolución del problema cinemático inverso

El problema cinemático inverso [4] consiste en resolver los valores que adoptarán cada una de las articulaciones para un posicionamiento particular del extremo del manipulador. Este problema es mucho más complejo de resolver que el cinemático directo ya que se trata de una función multivaluada altamente no lineal. Asimismo, este problema representa la dificultad ideal para abordar su resolución mediante redes neuronales artificiales.

3.1 Arquitectura de la Red Neuronal y Conjunto de Entrenamiento

No existen un algoritmo o reglas que permitan definir la arquitectura de una red neuronal para un problema determinado. En otras palabras no existe ningún tipo de teoría que explique el porqué de una u otra arquitectura para la resolución de cierto problema. Lo que se debe hacer es probar el comportamiento de distintas arquitecturas hasta encontrar aquella que tiene un desempeño aceptable en la resolución del problema abordado [5]. La red neuronal a emplear es del tipo Backpropagation [6]. Al ser esta una red supervisada, es necesario contar con un conjunto de entrenamiento.

En el conjunto de entrenamiento, los puntos a utilizar deben ser representativos del problema que se intenta solucionar. Esto es, si se utilizan menos puntos de los necesarios, la red será incapaz de aprender. Y en el caso de contar con un conjunto de entrenamiento con demasiados puntos, la red necesitará un tiempo excesivo para su entrenamiento y en el peor de los casos, su entrenamiento no convergerá.

La red neuronal empleada es del tipo Backpropagation y consta de tres capas ocultas con funciones de transferencia sigmoidales [7]. La primera capa posee 35 neuronas, la siguiente 25 neuronas, luego una capa de 15 neuronas y por último una capa de salida con una única neurona cuya función de transferencia es lineal [8]. La red admite como entrada la posición deseada del manipulador (X,Y,Z) y entrega como salida cada uno de los ángulos requeridos en las articulaciones (Q1, Q2, Q3) a efectos de alcanzar la posición deseada. Como se tiene una red por ángulo, se requieren tres redes neuronales (con arquitectura idéntica a la descripta) para definir los ángulos de desviación requeridos en las primeras tres articulaciones del robot. A efectos de facilitar el entrenamiento, el espacio de trabajo se ha dividido en ocho regiones iguales. Esto se debe a que el alto nivel de no linealidad del problema (si se considera el espacio completo) impide resolverlo mediante una única red neuronal. Dado que se cuenta con tres redes neuronales por cada uno de los sectores en que se ha dividido el espacio de trabajo, esto totaliza 24 redes neuronales.

Los puntos en el espacio de trabajo no están uniformemente distribuidos. Por el contrario, la densidad de puntos es mayor en la periferia de cada región. Esto se debe a que a la red le cuesta más generalizar en la frontera de las regiones. Teniendo en cuenta este comportamiento, lo que se decide es presentarle una mayor cantidad de puntos en estas zonas, de modo tal de aumentar la información referida a los límites de las zonas con las que se entrena la red. A tal efecto, en el software *Armbot* existe un entorno denominado *Entrenamiento*, el cual permite verificar la estructura del entrenamiento, así como la evolución del entrenamiento para cada una de las redes neuronales que resuelven el problema cinemático inverso en cada una de las ocho regiones en que se ha dividido el espacio de trabajo. A tal efecto, la interfase mostrada por el software se ilustra en la Figura 8.

Mediante la caja de selección de octeto (marcada como 1), se puede elegir una de las ocho regiones en que fue dividido el espacio de trabajo. Mediante la caja de selección de red (marcada como 2), se selecciona una de las tres redes neuronales asociadas a cada octeto. Una vez seleccionado el octeto y la red, es posible mediante una de las opciones ofrecidas por el menú contenido en la interfase, observar los puntos con los que ha sido entrenada la red, el propio entrenamiento de la red y verificar la construcción de este conjunto de entrenamiento. En la Figura 9 puede observarse el entrenamiento para el octeto número 4. En la misma, puede apreciarse la distribución no uniforme de los puntos de dicho conjunto.

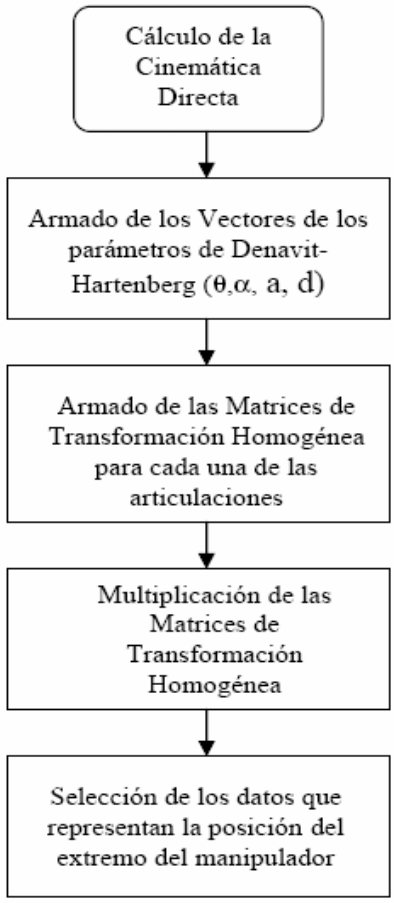


Figura 6. Secuencia de Pasos para el cálculo de la Cinemática Directa

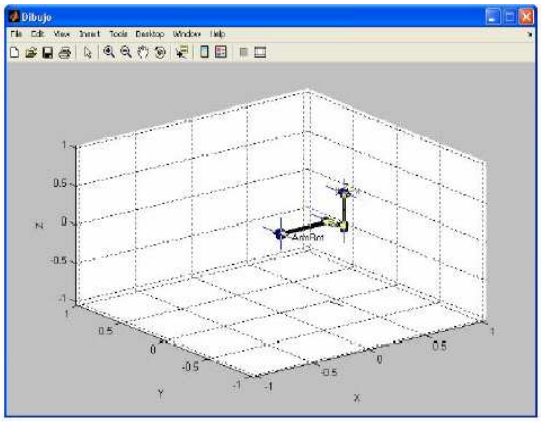


Figura 7. Gráfico de la posición del manipulador en el espacio

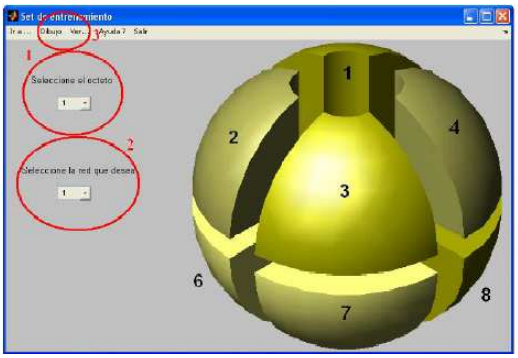


Figura 8. Interfase para el entrenamiento de las redes neuronales

3.2 Resolución del Problema Cinemático Inverso

En esta etapa de solución del problema, lo que se posee son las redes neuronales entrenadas. En esta instancia se utiliza un programa diseñado para que actúe como maestro de redes. Al ingresarle el punto en el espacio deseado para la ubicación del extremo del manipulador, expresado en coordenadas cartesianas, el mismo entrega como salida la región a la que pertenece dicho punto.

Para cada región del espacio se han entrenado tres redes neuronales, una por el ángulo de desviación correspondiente a cada articulación, tal como se explicara previamente. Este proceso de solución puede apreciarse a partir de la representación gráfica de la Figura 10. En la Figura 11 puede verse la pantalla de interfase del software que entrega como salida los ángulos Q1, Q2, Q3 como respuesta a la selección interactiva de un punto en el espacio de trabajo del manipulador. Asimismo, el software *Armbot* posee internamente un generador de trayectorias rectas para el robot. El generador de trayectorias divide la recta real y continua que une los puntos seleccionados y calcula una serie de puntos igualmente espaciados que componen la recta deseada. Luego, dichos puntos se ingresan en el software de resolución del problema cinemático inverso y se obtienen los ángulos de las articulaciones correspondientes a dichos puntos. De este modo, sólo resta graficar estos puntos en su correspondiente secuencia y se logra que el manipulador una dos puntos del espacio de trabajo mediante una recta. El software *Armbot* permite calcular el error porcentual, es decir la distancia euclidea entre los puntos de la recta real y de la recta obtenida. El gráfico que se obtiene es como el ilustrado en la Figura 12. Como puede apreciarse en el gráfico, al utilizar las redes neuronales propuestas para resolver el problema de la cinemática inversa, se consigue un error inferior al 0.5 %. Cabe aclarar que los experimentos fueron realizados en un computador del tipo PC Intel Pentium IV, 3 GHz, bajo sistema operativo Microsoft Windows XP Professional.

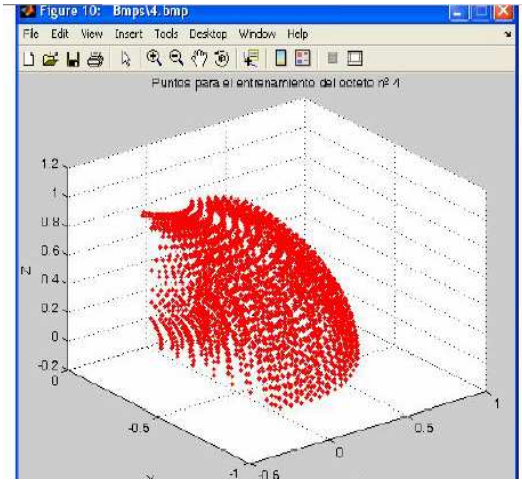


Figura 9. Visualización del conjunto de entrenamiento para el octeto Nº 4.

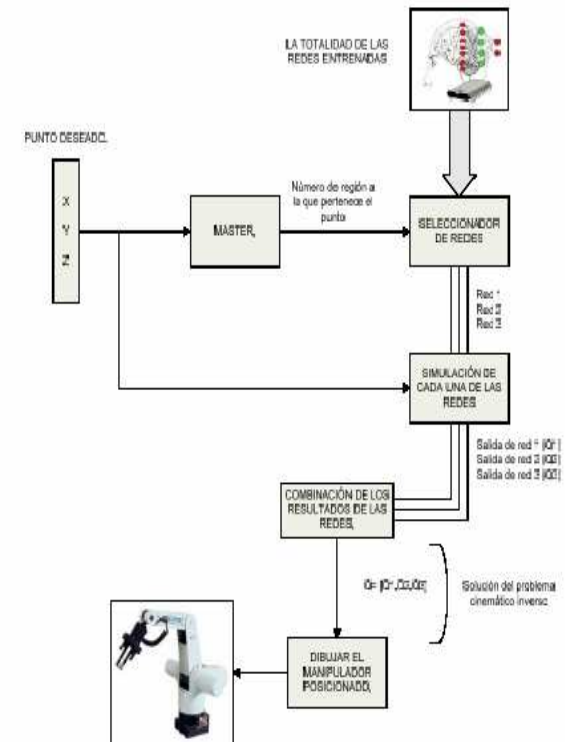


Figura 10. Proceso de obtención de la cinemática inversa del robot

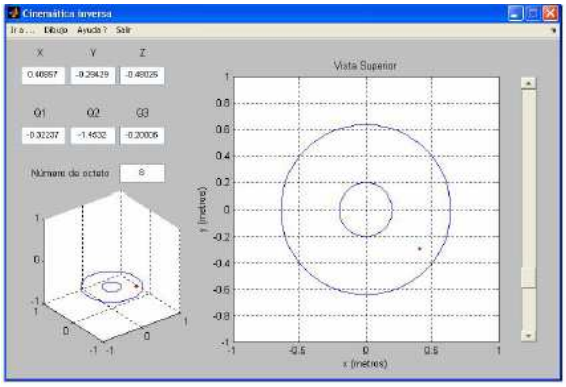


Figura 11. Punto seleccionado para la resolución

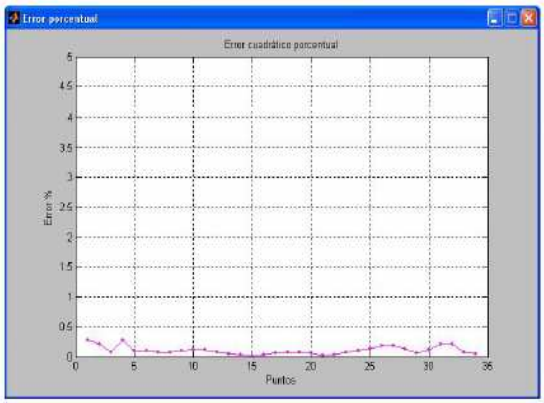


Figura 12. Gráfico del error

4. Conclusiones

En el presente trabajo se han resuelto los problemas cinemáticos de un manipulador robótico, tanto en forma directa como inversa. Dentro del ambiente de la robótica, el problema directo es de solución relativamente simple. Sin embargo, esto no ocurre con el problema cinemático inverso, el cual es no lineal y multivaluado. Por lo tanto, el empleo de redes neuronales para la resolución del problema cinemático inverso posee ciertos aspectos positivos, a saber:

- La cantidad de operaciones que debe realizar un procesador es siempre la misma (del orden de 13.200). Esto es, independientemente del punto seleccionado, la cantidad de operaciones es la misma. Esto se debe a que las únicas operaciones que se realizan son las necesarias para la simulación de la red ante una nueva entrada.
- No sólo la cantidad de operaciones es constante, sino que se trata de una cifra mucho menor que en los métodos heurísticos tradicionales. Evidentemente que la cantidad de operaciones dependerá de la arquitectura de la red neuronal seleccionada.
- La red neuronal, al igual que el cerebro humano no es proclive a confundir ruido con datos. Por lo tanto, el sistema es menos sensible al mismo.
- Asimismo, el empleo de redes neuronales en la resolución del problema abordado puede tener ciertos aspectos negativos tales como:
- Teniendo en cuenta la abundancia de información a ser procesada, el tiempo computacional requerido para el entrenamiento de las redes neuronales es elevado.
- El tiempo computacional necesario durante las fases de diseño y puesta a punto de las redes neuronales es elevado. Esto es, como se mencionó anteriormente, se debe experimentar con diferentes arquitecturas de red y conjuntos de entrenamiento.
- Debido a la dificultad del problema cinemático inverso, el conjunto de entrenamiento debió ser dividido en regiones. Esto muestra que cuando se trata de resolver un problema altamente no lineal, el tiempo de entrenamiento crece exponencialmente y el posterior comportamiento de las redes no es el deseado dado que no se logran errores de entrenamiento suficientemente bajos.

Asimismo, el empleo de métodos heurísticos tradicionales posee ciertas desventajas, tales como:

- En muchas aplicaciones el problema cinemático inverso debe resolverse en tiempo real (por ejemplo, en el seguimiento de una determinada trayectoria). Una solución de tipo iterativo no garantiza disponer de la solución en el momento adecuado.
- La solución al problema cinemático inverso puede poseer redundancia, es decir, existir más de una solución a la transformación cinemática inversa.
- En particular, si el número de soluciones es mayor que el número de variables de articulación, implica que el manipulador puede no tener solución para su cinemática inversa, por lo que se hace necesario imponer restricciones.
- La existencia de ciertas funciones matemáticas en la solución analítica a la transformación cinemática inversa, indica la existencia de redundancias (por ejemplo, la raíz cuadrada y el coseno no discriminan el signo).

- La eficiencia del modelo cinemático de un manipulador se mide usualmente en función del número de operaciones de adición, producto y llamadas a funciones trascendentes requeridas para resolver el modelo.

Dado que las de arquitecturas empleadas en este trabajo son del tipo off-line [9], [10], a pesar de los elevados tiempos de entrenamiento, una vez encontrados los pesos de la red, éstos no se modifican y por lo tanto no se precisa reentrenarlas (a menos que cambien las características del problema a resolver) y tan sólo simularlas.
Si bien las dificultades mencionadas ponen en una disyuntiva al diseñador en cuanto a la utilización de redes neuronales para la solución de problemas complejos, los resultados obtenidos en el presente trabajo son una demostración de la potencialidad de las mismas para la resolución de problemas específicos de alta complejidad.

5. Referencias

[1] Barrientos A., Peñin L., Balaguer C., Gy Aracil, R., *Fundamentos de Robótica*. Mac Graw Hill Madrid. 1997.

[2] Mc Kerrow, P. J. *Introduction to Robotics*. Addison Wesley. Sidney. 1998.

[3] Rentería, A., Rivas M., *Robótica industrial: fundamentos y Aplicaciones*. Mac Graw Hill. Madrid.2000.

[4] Ollero Baturone A., *Robótica: manipuladores y robots móviles*. Marcombo Boixareu Editores. Barcelona. 2001.

[5] García Martínez R., Servente M., Pasquini D., *Sistemas Inteligentes*. Nueva Librería. Buenos Aires. 2003.

[6] Hilera J.R., Martínez, V.J., *Redes Neuronales Artificiales: Fundamentos, Modelos y Aplicaciones*. Editorial Rama. Madrid. 1995.

[7] Hetch –Nielsen, R., *Neurocomputing: picking the human brain*. IEEE Spectrum. March, 1988.

[8] Feldman J. A., Ballard D. H., *Conexionist models and its properties*. Cognitive Science (6):205-254. 1982.

[9] Caudill M., Butler Ch., *Understanding Neural Networks: Computer explorations*. MIT Press. 1992.

[10] Sierra, E., García-Martínez, R., Hossian, A., Britos, P. y Balbuena, E. 2006. *Providing Intelligent User-Adapted Control Strategies in Building Environments*. Research in Computing Science Journal, 19: 235-241.

Estudio, Implantación y Resultados de la Adaptación Espacio Europeo de Educación Superior en las Asignaturas de Programación de la Titulación de Informática de la Universidad de Málaga

Belmonte M.V., Cotta C., Fernández A. J., Pastrana J. L., Soler E., YagüeM.I.
ETSI Informática. Campus Teatinos s/n 29071 Málaga. España
{mavi, ccottap, afdez, pastrana, esc, yague}@lcc.uma.es

Abstract

Las asignaturas de Elementos de Programación (EP), Laboratorio de Programación (LP) y Metodología de la Programación (MP) constituyen la base sobre la cual se explican los fundamentos básicos de la programación en la Escuela Técnica Superior de Ingeniería en Informática (ETSII) de Málaga. Actualmente la Universidad española está inmersa en un proceso de cambio motivado por la adaptación de los estudios universitarios al Espacio Europeo de Educación Superior (EEES). El objetivo de nuestro proyecto es adoptar el sistema de transferencia europeo de créditos en dichas asignaturas y de experimentar con las futuras formas de enseñanza (tanto docentes como evaluadoras). A lo largo del trabajo se mostrará cómo se ha realizado dicha adaptación, los problemas y soluciones adoptados, los sistemas de evaluación, las guía docentes elaboradas y los resultados obtenidos en el curso académico 2004/2005 y 2005/2006, así como ideas y trabajos futuros a seguir en esta línea.

1. Introducción

Entre los objetivos fundamentales de la Unión Europea se encuentra la coordinación de las políticas y normas legislativas de sus estados miembros en cuestiones relacionadas no sólo con el desarrollo económico, sino también con el progreso y el bienestar social de los ciudadanos. Este objetivo se ha extendido, en la última década, al ámbito de la educación y, muy singularmente, de la enseñanza superior, en el que diversos países, en un proceso que no hará sino incrementarse, han adoptado ya medidas conducentes a la reforma de la estructura y organización de sus enseñanzas universitarias para favorecer la construcción del Espacio Europeo de Educación Superior. En este proceso han desempeñado un papel decisivo las redes de colaboración existentes entre las instituciones universitarias europeas, el desarrollo de programas de movilidad interuniversitaria, y el impulso generado por las declaraciones tanto de los responsables académicos de estas instituciones como por los ministros de educación de los países que conforman la Unión Europea (Sorbona[01], Bolonia[02] y Praga).

La declaración de La Sorbona (1998), en la que aparece por primera vez el concepto de Espacio Europeo de Educación Superior, pone de manifiesto una voluntad decidida de potenciar una Europa del conocimiento de acuerdo con las tendencias que predominan en los países más avanzados socialmente, en los que la extensión y calidad de la educación superior son factores decisivos en el incremento de la calidad de vida de los ciudadanos. El Espacio Europeo de Enseñanza Superior significa un reto muy positivo para todos. Los estudios tendrán mayor transparencia y serán más fácilmente comparables con beneficios para toda la sociedad y reportará a los estudiantes la organización de las enseñanzas en función de su aprendizaje. La introducción del crédito europeo como unidad del haber académico valora el volumen global de trabajo realizado por el alumno en sus estudios, no sólo las horas de clase. El diseño de los planes de estudio y las programaciones docentes se llevarían a cabo teniendo como eje de referencia el propio aprendizaje de los alumnos.

El suplemento europeo al título ayudaría al reconocimiento más fácil y transparente por parte de otras universidades y organismos europeos de la formación adquirida. Y, en fin, la estructura de las enseñanzas que cursan y los niveles de los títulos que reciben al finalizar sus estudios serían más homogéneos con los correspondientes títulos y enseñanzas de los países de la Unión Europea

favoreciendo su movilidad e integración en el mercado laboral. Tan importante como el objetivo compartido de una armonización de los diversos sistemas que regulan las enseñanzas universitarias en cada estado miembro es la convicción, conjuntamente asumida, de que este proceso ha de llevarse a cabo con la máxima colaboración y participación de las propias instituciones de enseñanza superior así como con el máximo respeto a la diversidad de culturas y a la autonomía universitaria.

Dentro del proceso de Convergencia al Espacio Europeo de Educación Superior (EEES) [03] que constituirá una reforma del sistema educativo universitario español, y que tiene como objetivos fundamentales el aumento de la calidad de la docencia y el fomento de la movilidad estudiantil entre países de la Unión Europea. La Convergencia al EEES se está materializando a través de iniciativas legislativas de los distingos gobiernos (nacional y de las Comunidades Autónomas), y en la realización, por parte de los centros universitarios, de estudios y experiencias previos sobre el proceso.

Por iniciativa de la Dirección de la E.T.S.I. Informática de la Universidad de Málaga, y con el acuerdo de la Junta de Centro en reunión celebrada el 24 de Junio de 2004, durante el curso 2004-2005, uno de los grupos de primer curso de la Ingeniería Técnica de Informática de Gestión adecuó todas sus asignaturas al Sistema Europeo de Transferencia de Créditos (ECTS). Esto supuso una modificación de las prácticas docentes que el profesorado de dichas asignaturas viene desempeñando en los últimos años, estableciendo nuevas formas de enseñanza, basadas en la participación activa del alumno frente a la exclusividad de la clase magistral, y nuevos métodos de evaluación, que deberán estar fundamentados en el aprovechamiento del esfuerzo realizado por el estudiante, y no en la cantidad de contenidos que el profesor haya sido capaz de transmitir. El éxito de esta experiencia piloto ha hecho que en el curso 2005-2006 todos los grupos de primer curso de las tres titulaciones de informática de la Universidad de Málaga hayan adecuado sus asignaturas a este nuevo sistema.

2. Asignaturas de Programación en la ETSI Informática

El objetivo global del curso de programación de primero es proporcionar al alumno los elementos básicos, de forma rigurosa y estructurada, haciendo especial hincapié en el seguimiento de una metodología de programación. Al mismo tiempo se proporcionará al alumno algunos conceptos básicos sobre eficiencia, complejidad de algoritmos y verificación formal a nivel muy elemental.

El alumno deberá saber resolver problemas de programación usando las herramientas habituales que se exigen a un programador, y deberá de plantear sus soluciones de forma abstracta para posteriormente ir al detalle, siguiendo una programación modular y estructurada en las ocasiones en las que el tamaño del problema lo requiera, y definiendo las estructuras de datos adecuadas para su resolución.

Las asignaturas objetos de este trabajo son las que deben de sentar las bases para posteriores asignaturas de programación en otros cursos, por lo que deben de tratarse su elaboración con sumo cuidado, pues debemos de preparar al alumno para facilitarle una posterior ampliación y perfeccionamiento de conceptos, así como evitar que adopten metodologías incorrectas y vicios que después son difíciles de corregir.

Para cumplir estos objetivos, no consideramos necesario el aprendizaje de un lenguaje de programación concreto. Se opta por utilizar un pseudolenguaje que aporte una notación sencilla, bien definida, pero sin estar afectado por diferentes criterios de implementación ni sujeto a restricciones sintácticas demasiado rigurosas. De esta forma, nos centramos precisamente en la tarea de diseño de algoritmos, olvidándonos detalles concretos que introduciría un lenguaje de programación. Con ello también se le hace ver al alumno que los conocimientos que irá adquiriendo a lo largo del curso son independientes de cualquier lenguaje de programación, separando el diseño de algoritmos de su codificación en un lenguaje determinado.

El alumno deberá adquirir conceptos de programación modular, aplicarlos, implementar bajo diferentes formas la solución a un problema caracterizando cada una de las implementaciones, analizar la complejidad de los algoritmos a nivel elemental, resolver problemas mediante algoritmos recursivos sabiendo identificar las ventajas e inconvenientes de esta técnica, así como decidir cuando será necesario usarla, llegar a dominar los conceptos relacionados con las estructuras de datos más complejas, conocer perfectamente determinadas estructuras lineales y en árbol, decidir cuando interesa usarlas, y comprender las limitaciones de la algoritmia ante cierta clase de problemas.

Para llevar a cabo un curso de programación completo, debemos de proporcionar al alumno las herramientas necesarias para poner en práctica los conocimientos que va adquiriendo con las diferentes asignaturas de programación. Para ello, estas asignaturas están apoyadas en cada cuatrimestre por unas prácticas de laboratorio en el que los alumnos toman contacto con un ordenador y realizan prácticas en él, resolviendo problemas planteados por el profesor en un determinado lenguaje de programación.

Uno de los objetivos principales de las asignaturas de Laboratorio es que los alumnos contacten con el entorno de la máquina, conozcan y manejen sus componentes, y aprendan los aspectos y comandos básicos del sistema operativo. Además, el alumno aprenderá un lenguaje de programación en concreto que deberá de usar para resolver los problemas planteados por el profesor, siguiendo los criterios metodológicos estudiados en las asignaturas de Elementos de Programación. Para ello se deberá de desenvolver adecuadamente en un entorno de programación con el que codificará, depurará, compilará y ejecutará los programas, adquiriendo alguna práctica y soltura en este tipo de entornos, actualmente muy comunes en el campo de la programación.

Las asignaturas de programación de primer curso en la ETSI Informática de Málaga tratan de proporcionar a los estudiantes los conceptos y estructuras propios del paradigma de programación imperativa. Estas asignaturas son medidas en créditos LRU (Ley de Reforma Universitaria), cuya medida corresponde a 1 crédito = 10 horas de clase magistral por parte del profesor. Por tanto, la carga docente de las asignaturas es:

Asignatura	LRU	Horas/Semana
EP	7.5	5
MP	6	4
LP	4.5	3

Tabla 1. Créditos/Horas de las Asignaturas

3. El Crédito ECTS (European Credit Transfer System)

El crédito ECTS es una nueva medida del peso de una asignatura basada en el trabajo del alumno en vez de basarse en el número de horas de clase magistral del profesor. En esta unidad de medida se integran:

- Enseñanzas teóricas.
- Enseñanzas prácticas.
- Actividades académicas dirigidas (biblioteca, Internet, exámenes, trabajo de campo, tutorización, etc.).
- Horas de estudios y de trabajo del estudiante

La principal finalidad de esta medida (además de tener en cuanta el trabajo del alumno) es el reconocimiento de los estudios y de los títulos como condición previa para la creación de un espacio europeo abierto en materia de educación y formación en el que los estudiantes y los profesores puedan desplazarse sin obstáculos. Los objetivos de uso de créditos ECTS son:

- la utilización de créditos ECTS como valores que representan el volumen de trabajo efectivo del estudiante (workload) y el rendimiento obtenido mediante calificaciones comparables (ECTS grades).
- la información sobre los programas de estudios y los resultados de los estudiantes con documentos con un formato normalizado (Guía docente y certificados académicos).

En España se aplicará en las directrices generales propias correspondientes a títulos universitarios de carácter oficial con obligatoriedad a partir de 2010/2013.

4. Descripción de la experiencia

La experiencia ha consistido básicamente en implantar una metodología de enseñanza basada en evaluar el trabajo que el alumno debe realizar para aprobar la asignatura; se debe valorar pues no sólo la nota obtenida en el examen final, sino además el trabajo realizado fuera de las clases (estudio, realización de problemas y ejercicios en casa, , asistencia a las clases, etc). Para ello implantamos una nueva metodología consistente en la evaluación periódica de los conocimientos que los alumnos van adquiriendo paulatinamente a lo largo del curso y en la valoración de trabajos que el alumno podía realizar voluntariamente fuera del contexto de la clase magistral. Impusimos asimismo un mecanismo de evaluación basado en pesos que primaba la asistencia a clase y a tutorías, la realización de trabajos, las notas obtenidas en los controles de conocimiento y la nota obtenida en el examen.

Para la realización de la experiencia piloto se han realizado numerosas reuniones entre los profesores involucrados en el proyecto y entre los profesores que imparten docencia, no sólo en asignaturas similares,

sino en otras asignaturas correspondientes al plan de 1ª de ingeniería técnica en informática. Se debe tener en cuenta que el contexto en el cual se aplica el proyecto de innovación educativa engloba a todos los grupos de primer curso de la titulación de la Ingeniería Técnica en Informática de Gestión y de Sistemas (7 grupos en total). Se realizaron numerosas reuniones organizadas, tanto a nivel interno (esto es, entre los profesores involucrados en el proyecto) como a nivel general (i.e., con los profesores que imparten docencia en los cursos involucrados y con la presencia del Subdirector de Ordenación Académica).

En líneas generales esas reuniones se han destinado a: estudiar la transformación de actuales créditos LRU en créditos ETCS (i.e., el denominado crédito europeo, European Transfer Credit System) para las asignaturas de programación, estudiar la carga docente del profesor en las asignaturas EP/LP/MP según créditos ETCS, estudiar la planificación semanal y carga de trabajo del alumno, preparar las primeras propuestas completas de guías docente, consensuar la traducción de créditos LRU a créditos ETCS en todas las asignaturas de los grupos involucrados en el proyecto, preparar las propuestas de temporización, discutir las propuestas comunes sobre criterios de evaluación y seguimiento, unificar los criterios de evaluación en todas las asignaturas involucradas en el proyecto, determinar el conjunto de trabajos particulares para cada alumno de los grupos de EP, MP y LP involucrados en el proyecto y el sistema de evaluación de los mismos, desarrollar los test parciales a realizar por el alumno, analizar los resultados estadísticos obtenidos en el grupo piloto y comparación con el resto de grupos que siguen el sistema de enseñanza vigente en la ETSII y valorar estadísticamente los resultados obtenidos por la aplicación de la experiencia piloto en el primer cuatrimestre en la asignatura de EP (ya que MP y EP están siendo evaluadas en el momento de escribir este trabajo). Adicionalmente, para cada asignatura se ha diseñado una guía docente adaptada al sistema de créditos europeos y al EEES. Esta guía docente ha sido seguida a lo largo del curso.

5. Guías Docentes

A modo de Ejemplo mostraremos la guía docente y su elaboración de la asignatura de Elementos de Programación y estaremos encantados de mandar las otras guías a cualquier lector que lo solicite.

La elaboración de la guía docente de una asignatura que siga la filosofía del EEES requiere dos tareas fundamentales: identificar las competencias requeridas y a adquirir en la asignatura [04,05] (tanto cognitivas como transversales relativas a las competencias personales, instrumentales y sistémicas), así como al cálculo del trabajo del alumno y la temporalización semanal del mismo, de modo que el trabajo total del alumno, sumando todas las asignaturas de un curso completo, no supere las 40 horas semanales, que es lo que se considera que debe trabajar como máximo un trabajador en la Unión Europea.

5.1. Capacidades del título de grado para Ingeniería en Informática

Las personas que han obtenido el título de Ingeniería en Informática son profesionales con una formación amplia y sólida que les prepara para dirigir y realizar las tareas de todas las fases del ciclo de vida de sistemas, aplicaciones y productos que resuelvan problemas de cualquier ámbito de las Tecnologías de la Información y las Comunicaciones, aplicando su conocimiento científico y los métodos y técnicas propios de la ingeniería.

Con carácter general, el Ingeniero en Informática está capacitado para aprender a conocer, hacer, convivir y ser, en su ámbito personal, profesional y social, de acuerdo con lo recogido en el informe de la UNESCO sobre las perspectivas de la educación en el siglo XXI. Por su formación, tanto en su base científica como tecnológica, las personas tituladas en Ingeniería en Informática se caracterizan por:

- Estar preparadas para ejercer la profesión, teniendo una conciencia clara de su dimensión humana, económica, social, legal y ética.
- Estar preparadas para, a lo largo de su carrera profesional, asumir tareas de responsabilidad en las organizaciones, tanto de contenido técnico como directivo, y de contribuir en la gestión de la información y en la gestión del conocimiento.
- Tener las capacidades requeridas en la práctica profesional de la ingeniería: ser capaces de dirigir proyectos, de comunicarse de forma clara y efectiva, de trabajar en y conducir equipos multidisciplinares, de adaptarse a los cambios y de aprender autónomamente a lo largo de la vida.
- Estar preparados para aprender y utilizar de forma efectiva técnicas y herramientas que surjan en el futuro. Esta versatilidad les hace especialmente valiosos en organizaciones en

las que sea necesaria una innovación permanente.
- Ser capaces de especificar, diseñar, construir, implantar, verificar, auditar, evaluar y mantener sistemas informáticos que respondan a las necesidades de sus usuarios.
- Tener la formación de base suficiente para poder continuar estudios, nacionales o internacionales, de Máster y Doctorado..

5.2. Cálculo de la Carga Docente

Para calcular los créditos ECTS que corresponde a una asignatura debemos calcular el número de horas presenciales (en clase) y no presenciales (no autorizadas) que requiere un alumno medio para ser capaz de superar el temario de la asignatura. Para ello debemos analizar tema a tema, el número de horas teóricas, prácticas y de laboratorio y luego aplicar la relación de 30 horas por crédito ECTS (T = Teoría, P = Problemas y L = Laboratorio).

- LRU: 7.5 = 4.5 (T) + 1.5 (P) + 1.5 (L) = 75 horas presenciales
- ECTS: 6.0 = (6.0 ECTS x 30horas/ECTS = 180 horas de trabajo del estudiante)

Tema	**Horas Presenciales**			**Horas No Presenciales**		
	T	*P*	*L*	*T*	*P*	*L*
1	1			4		
2	2	1		4	4	
3	2	1		4	4	
4	1		2	3		8
5	4	4	4	6	12	12
6	2	3	2	4	10	8
7	5	4	2	10	12	9
Total	17	13	10	35	42	37
Examen	4			20		
Tests	2					
Total	46			134		

Tabla 2. Relación de Horas

Carga semanal según creditos ECTS
180 horas / 22.5 semanas = 8 horas/semana

5.3. Un Ejemplo de Guía Docente: Elementos de Programación

A partir de los estudios realizados en los apartados anteriores, se elabora una guía docente de la asignatura que es aprobada por consejo de departamento y junta de centro y que se facilita a los alumnos en el libro de matriculación y también es pública electrónicamente.

DATOS ESPECÍFICOS DE LA ASIGNATURA

1. Descriptores de la asignatura:

Propedéutica al diseño de algoritmos y de programas.

2. Situación de la asignatura.

2.1. Prerrequisitos:

Ninguno

2.2. Contexto dentro de la titulación:

Esta asignatura debe aportar al alumno los conocimientos de programación básicos, de una manera rigurosa y estructurada. Además, debe servir de base para el resto de las asignaturas de programación de la titulación.

2.3. Recomendaciones:

Se recomienda que el alumno tenga conocimientos básicos de matemáticas y lógica.

3. Competencias a adquirir por los estudiantes. (A = Alto, M = Medio, B = Bajo)

3.1. Competencias transversales o genéricas.

3.1.1. Competencias instrumentales:

A M B
☒ ☐ ☐ Capacidad de análisis y síntesis.
☒ ☐ ☐ Capacidad de organización y planificación.
☒ ☐ ☐ Comunicación oral y escrita.
☐ ☒ ☐ Conocimiento de una lengua extranjera.
☐ ☒ ☐ Capacidad de gestión de la información.
☒ ☐ ☐ Resolución de problemas.
☐ ☒ ☐ Toma de decisiones.
☐ ☒ ☐ Capacidad de Abstracción
☐ ☒ ☐ Capacidad para relacionar conceptos

3.1.2. Competencias personales:

A M B
☐ ☒ ☐ Trabajo en equipo.
☐ ☐ ☒ Habilidades en relaciones

interpersonales.
☐ ☐ ☒ Capacidad para comunicarse con expertos.
☒ ☐ ☐ Razonamiento crítico.

3.1.3. Competencias sistémicas:

A M B
☐ ☒ ☐ Aprendizaje autónomo.
☐ ☒ ☐ Adaptación a nuevas situaciones.
☒ ☐ ☐ Capacidad de aplicar los conocimientos.
☒ ☐ ☐ Habilidad para trabajar de forma autónoma.
☒ ☐ ☐ Creatividad.
☐ ☐ ☒ Liderazgo.
☐ ☒ ☐ Iniciativa y espíritu emprendedor.
☐ ☒ ☐ Motivación por la calidad.

3.2. Competencias específicas.

3.2.1. Competencias cognitivas (saber):

Conocimiento y comprensión: Demuestra conocimiento y comprensión de hechos esenciales, conceptos, principios y teorías relacionados con la informática y las aplicaciones software.
Requisitos. Identifica y analiza los criterios y características técnicas apropiadas a problemas específicos, así como las estrategias para planificar su solución.
Métodos y herramientas. Hace uso de las teorías, prácticas y herramientas apropiadas para la especificación, planificación, aplicación y evaluación de sistemas software.

3.2.2. Competencias procedimentales e instrumentales (saber hacer):

A M B
☒ ☐ ☐ Planteamiento de soluciones algorítmicas.
☐ ☒ ☐ Resolución de modelos utilizando técnicas analíticas, numéricas, estadísticas.
☐ ☒ ☐ Visualización/interpretación de soluciones.
☒ ☐ ☐ Implementación programas informáticos.
☒ ☐ ☐ Diseño e implementación de algoritmos.
☒ ☐ ☐ Identificación y localización de errores.
☐ ☒ ☐ Argumentación lógica: toma de decisiones.
☒ ☐ ☐ Capacidad de conectar teoría y práctica.
☒ ☐ ☐ Diseño de experimentos y estrategias.
☒ ☐ ☐ Utilización de herramientas.
☐ ☒ ☐ Participación en la organización y dirección de proyectos.

3.2.2. Competencias actitudinales (ser):

A M B
☐ ☒ ☐ Conocimiento de los procesos de aprendizaje de la informática.
☐ ☒ ☐ Aplicación de la informática a otras disciplinas y problemas reales.
☐ ☐ ☒ Capacidad de mostrar la vertiente lúdica de la informática.
☐ ☒ ☐ Expresión rigurosa y clara.
☐ ☒ ☐ Capacidad de comunicación oral y escrita.
☒ ☐ ☐ Capacidad de presentación de soluciones.
☐ ☒ ☐ Razonamiento lógico e identificación de errores en los procedimientos.
☒ ☐ ☐ Interés por la informática y aplicaciones.
☐ ☒ ☐ Relacionar informática y otras disciplinas.
☐ ☒ ☐ Capacidad actualización del conocimiento.
☐ ☒ ☐ Capacidad de crítica.
☐ ☒ ☐ Capacidad de adaptación.
☒ ☐ ☐ Capacidad de abstracción.
☐ ☐ ☒ Capacidad de estimación de necesidades.
☐ ☒ ☐ Capacidad de gestión y organización.

4. Objetivos:

Iniciar al alumno en las técnicas de programación estructurada, usando metodología descendente.

5. Metodología (en horas de trabajo del estudiante):

	Cuatrimestre	
	I	II
Presenciales		
Clases de teoría:	17,0	0,0
Clases de problemas:	25,0	0,0
Clases prácticas en laboratorio:	12,0	0,0
Seminarios y exposiciones:	0,0	0,0

Exámenes:	6,0	0,0
No presenciales		
Estudio de clases teóricas	34,0	0,0
Estudio de problemas y prácticas	74,0	0,0
Preparación de trabajos Individuales	12,0	0,0
Preparación de trabajos en grupos:	0,0	0,0
Total:	**120**	**0,0**
Trabajo total del estudiante: 180,0 horas.		
Presenciales: 60	**No presenciales 120**	

6. Técnicas docentes.

6.1. Técnicas docentes utilizadas:

- ☒ Sesiones académicas de teoría.
- ☒ Sesiones académicas de problemas.
- ☒ Sesiones prácticas en el laboratorio.
- ☒ Trabajo en grupos reducidos.
- ☒ Resolución y entrega de problemas/prácticas.
- ☒ Realización de pruebas parciales evaluables.

6.2. Desarrollo y justificación:

Las clases presenciales se llevarán a cabo utilizando pizarra y transparencias. Se facilitarán ejercicios y trabajos junto con bibliografía específica de cada tema para el trabajo no presencial del alumno. Se utilizará una notación algorítmica específica, haciendo hincapié en la separación entre el diseño de algoritmos y su codificación. Sin embargo, la necesidad de que el alumno lleve a la práctica los conocimientos que va adquiriendo hace imprescindible la realización de algunas prácticas de laboratorio sobre un entorno de programación concreto. La asignatura de segundo cuatrimestre *Laboratorio de Programación* avanzará más en este aspecto y estará dedicada por completo a prácticas de laboratorio.

7. Bloques temáticos:

I. Introducción y Conceptos Previos
  1. Introducción y Conceptos Básicos
  2. Resolución de Problemas y Algoritmos
  3. Lenguajes de Programación

II. Estructuras Simples de Programación
  4. Tipos de Datos Simples
  5. Estructuras de Control
  6. Subprogramas

III. Estructuras de Datos
  7. Tipos de Datos Estructurados

8. Temario desarrollado:

1. INTRODUCCION Y CONCEPTOS BÁSICOS
  - 1.1. Introducción a la Informática.
  - 1.2. El computador.
  - Herramientas básicas de un sistema.

2. RESOLUCION DE PROBLEMAS Y ALGORITMOS
  - 2.1. Resolución de problemas.
  - 2.2. Elementos metodológicos.

3. LENGUAJES DE PROGRAMACIÓN
  - 3.1. Introducción.
  - 3.2. Reconocimiento de lenguajes.
  - 3.3. Introducción al pseudolenguaje.

4. TIPOS DE DATOS SIMPLES
  - 4.1.- Elementos básicos.
  - 4.2. Clasificación de los tipos.
  - 4.3. Tipos simples predefinidos.
  - 4.4. Tipos simples definidos por usuario.

5. ESTRUCTURAS DE CONTROL
  - 5.1. Estructuras de selección.
  - 5.2. Estructuras de iteración.
  - 5.3. Prácticas de laboratorio

6. SUBPROGRAMAS
  - 6.1. Procedimientos y Funciones
  - 6.2. Anidamientos y ámbitos.
  - 6.3. Prácticas de laboratorio

7. TIPOS DE DATOS ESTRUCTURADOS
  - 7.1. Arrays.
  - 7.2. Cadenas de caracteres.
  - 7.3. Registros.
  - 7.4. Prácticas de laboratorio

9. Bibliografía.

9.1. Bibliografía general:

[DALE89a] DALE, N y WEEMS, C. Pascal. Segunda Edición. McGraw-Hill, 1989.

[JOYA96] JOYANES, L. Fundamentos de Programación. Algoritmos y Estructuras de Datos. McGraw-Hill, 1996.

[JOYA00] JOYANES, L. Programación en C++. Algoritmos, estructuras de datos y objetos. McGraw Hill, 2000.

[SAVI00] SAVICH, W. Resolución de problemas con C++. 2ª Edición. Prentice-Hall, 2000.

9.2. Bibliografía específica:

[JOYA96b] JOYANES, L. Fundamentos de Programación. Libro de problemas. McGraw-Hill, 1996.

[BROO95] BROOKSHEAR, J. G. Introducción a las Ciencias de la Computación. Cuarta Edición, Adisson-Wesley, 1995.
[WIRT80] WIRTH, N. Algoritmos + Estructuras de Datos = Programas. Ediciones del Castillo, 1980.

10. Técnicas de evaluación.

10.1. Técnicas de evaluación utilizadas:

☒ Examen teórico-práctico.
☒ Trabajos desarrollados durante el curso.
☒ Participación activa en las sesiones académicas.
☒ Controles periódicos de conocimientos.

10.2. Criterios de evaluación y calificación:

Se realizarán 4 controles a lo largo del cuatrimestre junto con un examen final. Se evaluará la asistencia a las clases presenciales así como la asistencia continuada a las tutorías (tanto las personalizadas como las comunes que se dictaminarán a lo largo del curso). Además se evaluará la realización y entrega de trabajos, así como la participación activa del alumno en la resolución de problemas que irán proponiéndose a lo largo del curso. La nota final de la asignatura se calculará de la siguiente manera:

- Asistencia clases presenciales y tutorías: **10%**
- Realización de trabajos propuestos:**5%**
- Resolución de problemas propuestos:**10%**
- Controles: **20% (4x5%)**
- Examen Final: **55%**

El alumno tendrá la calificación final de No Presentado únicamente si no se presenta al examen final y además a lo largo del curso se presenta a menos de cuatro de los controles propuestos**.**

6. Conclusiones y Trabajos Futuros

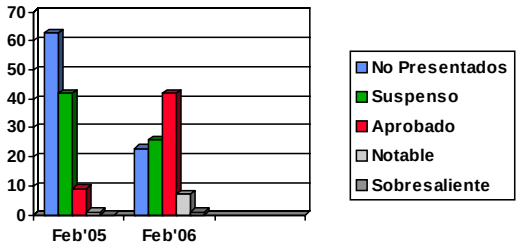
Podemos expresar con satisfacción que hemos alcanzado los objetivos planteados inicialmente. Se ha realizado una experiencia piloto durante el curso 2004/05 en un solo grupo y en 2005/06 con todos los grupos que nos ha permitido desarrollar las guías docentes de las asignaturas de programación para un mejor servicio a la paulatina adaptación al EEES. El grado de participación de los alumnos ha sido muy elevado y hemos tenido la gran satisfacción del aumento de la asistencia a clase, de alumnos presentados y de alumnos aprobados como puede verse en el análisis estadístico que se muestra a continuación con la comparativa de los dos últimos cursos en un grupo (A de Gestión) en la asignatura de Elementos de programación y en el que el curso 2004/05 seguía una metodología tradicional y en el curso 2005/06 sigue una metodología basada en el EEES.

	Acta		Presentados	
	Total	%	Total	%
NP	63	54,78	-	-
SUS	42	36,52	42	80,77
APR	9	7,83	9	17,31
NOT	1	0,87	1	1,92
SOB	0	0,00	0	0,00
	115		52	

Tabla 3. Convocatoria Febrero 2004/05

	Acta		Presentados	
	Total	%	Total	%
NP	23	23, 23	-	-
SUS	26	26,26	26	34,21
APR	42	42,42	42	55,26
NOT	7	7,07	7	9,21
SOB	1	1,01	1	1,32
	99		76	

Tabla 4. Convocatoria Febrero 2005/06



Valoramos la experiencia positivamente pero creemos que no tenemos recursos suficientes para su implantación total. Queremos seguir mejorando las técnicas docentes, las técnicas de evaluación y aumentando los recursos para los alumnos. Hay que trabajar en la mejora en la falta de recursos, información y formación del profesorado, colaboración de los alumnos, así como la mejora de los recursos (aulas e incremento del personal docente). El tema del incremento de personal, si bien no es competencia del personal docente, creemos que es de gran importancia, ya que el aumento de la carga trabajo del profesor para llevar un trato y evaluación más personalizada del

alumno es muy significativo y si el número de alumnos por grupo no es sustancialmente menor no va a ser factible la implementación real de la metodología a gran escala.

10. References

[01] Declaración de La Sorbona. Declaración conjunta para la armonización del diseño del Sistema de Educación Superior Europeo (a cargo de los cuatros ministros representantes de Francia, Alemania, Italia y el Reino Unido). La Sorbona, París, 25 de mayo de 1998, en http://ww.aneca.es/modal_eval/docs/declaracion_sorbona.pdf

[02] La Declaración de Bolonia, en http://www.aneca.es/modal_eval/docs/declaracion_bolonia.pdf

[03] Declaración de Bolonia: Adaptación del Sistema Universitario Español a sus Directrices, Conferencia de Rectores de Universidades Españolas., en http://www.crue.org/apadsisuniv.htm

[04] Computing Curricula 2001, Computer Science Volume, December 15, 2001, en http://www.sigcse.org/cc2001/

[05] C.Cotta, Antonio J.Fernández, J.Pastrana, E.Soler, M.I.Yagüe, PROYECTO ÁPICE3YSUMA: ADAPTACIÓN DE LA PROGRAMACIÓN EN INFORMÁTICA AL EEES EN LA UNIVERSIDAD DE MÁLAGA, en: Ángel Blanco López (Eds.), Innovación Educativa y Enseñanza Virtual en la Universidad de Málaga (Servicio de Innovación Educativa y el Servicio de Enseñanza Virtual y Laboratorios Tecnológicos, Málaga, 2005) 1-14.

Ontologías en el desarrollo de Entornos Virtuales para Entrenamiento

Raúl A. Aguilar
Universidad Autónoma de Yucatán
avera@uady.mx

Angélica de Antonio
Universidad Politécnica de Madrid
angelica@fi.upm.es

Fidel Rojas-Toledo
Universidad de Tarapacá
frojas@uta.cl

Resumen

Las ontologías representan una herramienta bastante útil para el desarrollo de sistemas software basados en conocimiento, como son los Entornos Virtuales Inteligentes. En este artículo se describe un par de ontologías que han sido diseñadas para la representación de conocimiento que es utilizado por una herramienta de planificación, la cual se encuentra integrada a un Entorno Virtual Colaborativo que sirve de apoyo al entrenamiento de equipos humanos. La arquitectura de dicha herramienta diseñada para la co-construcción de planes de actuación en tareas de naturaleza socio técnica, es también descrita.

1. Introducción

Existen diversas definiciones del término Ontología, las cuales se corresponden con su ámbito de aplicación; en Inteligencia Artificial (IA) por ejemplo, encontramos definiciones como la de Gruber [1] que la describe como una especificación explícita de una conceptualización. Noy y McGuinness [2] aportan una definición más constructiva, éstos la definen como una descripción explícita de conceptos en un dominio del discurso, con propiedades que describen sus características, atributos y restricciones del mismo.

El uso de ontologías en diversas disciplinas de las Ciencias Computacionales ha proliferado en los últimos años, y sus aplicaciones en la educación (Informática Educativa) no han sido la excepción. En este artículo se describe el diseño de ontologías que han sido elaboradas para su utilización como mecanismo de comunicación entre una herramienta de planificación —integrada a un Entorno Virtual Colaborativo (EVC) de apoyo al entrenamiento— y un Entorno Virtual Inteligente para Entrenamiento (EVIE). El artículo describe también la funcionalidad que debe ofrecer la herramienta de planificación en el EVC, como parte de una estrategia de entrenamiento previamente diseñada, así como su integración con el EVIE.

2. Entornos Virtuales para Entrenamiento

Una de las líneas de investigación que se viene desarrollando en el Laboratorio Decoroso Crespo de la Universidad Politécnica de Madrid (http://decoroso.ls.fi.upm.es/), es el de los Entornos Virtuales Inteligentes para la Formación y Entrenamiento; dichos entornos pueden ser utilizados para entrenar a uno o más estudiantes en la ejecución de ciertas tareas, particularmente, en aquellas situaciones en las que el entrenamiento en entornos reales es casi imposible o indeseable, por ser muy costoso o peligroso.

TEAM TRAIN es el nombre que le hemos asignado a uno de los proyectos derivados de ésta línea, su objetivo consiste en desarrollar una estrategia para entrenamiento de pequeños grupos asistida por Entornos Virtuales Inteligentes (EVI). La estrategia original descrita en [3] ha sido ampliada a cinco fases interrelacionadas (ver fig. 1.) en las que el grupo a entrenarse realiza un proceso iterativo de auto-evaluación en torno a la ejecución de una tarea —de naturaleza socio técnica— propuesta.

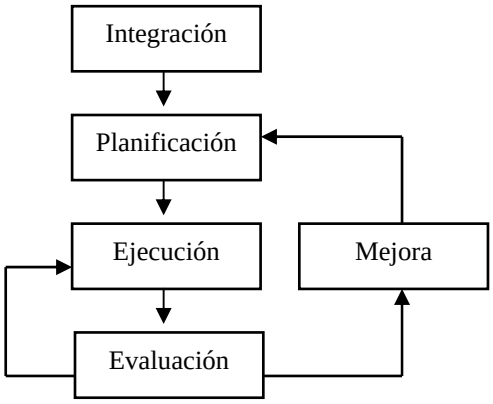


Figura 1: Fases de la Estrategia para Entrenamiento de Equipos

La fase de planificación, que ha sido añadida al esquema original de la estrategia de entrenamiento, plantea como dinámica para el equipo humano, la co-construcción de un plan de ejecución para la tarea que ha sido descrita en la fase anterior (Integración). Se propone que sea el propio equipo humano, el que durante una reunión virtual, diseñe —con una dinámica colaborativa— el plan que deberá utilizar durante su ejercitación en la siguiente fase (Ejecución). La fase de planificación ha sido incluida a la estrategia de entrenamiento con un doble propósito:

1. Refinar el esquema mental compartido del equipo humano en torno a la tarea por realizar.
2. Obtener, mediante un mecanismo de modelado, un primer modelo del grupo para el equipo humano, el cual será utilizado por el EVIE durante la fase de ejercitación.

La estrategia de entrenamiento considera la inclusión de un Entorno de Apoyo al Entrenamiento —de tipología bidimensional— para asistir al equipo en las fases de Integración, Planificación, Evaluación y Mejora, así como de un EVIE —con tipología 3D— que será utilizado para la ejercitación en la tarea durante la fase de Ejecución.

3. Una Herramienta Colaborativa para la Co-Construcción de Planes

Para realizar la dinámica mencionada en la fase de Planificación, el equipo investigador está implementando una herramienta que será integrada al área de trabajo (pizarra virtual) del Entorno de Apoyo al Entrenamiento previamente desarrollado para las fases de Integración y Evaluación [4], esta herramienta deberá mantener comunicación con el Entorno Virtual Inteligente desarrollado para el Entrenamiento, en particular, deberá de comunicarse con el Mundo Virtual para consultar, de los escenarios virtuales, las acciones posibles en el entorno, así como los objetos y actores que en éste se encuentren inmersos. La figura 2 ilustra la arquitectura propuesta para la herramienta de planificación, así como sus interrelaciones tanto con el EVI como con el equipo humano. La herramienta de planificación está integrada básicamente tres componentes que son descritos a continuación:

A. Editor. Es utilizado por el equipo humano durante una reunión virtual colaborativa, para la elaboración de un plan de ejecución. La herramienta realiza consultas a la Ontología del Mundo, tanto para la carga de los valores posibles de las acciones y sus argumentos, como para la validación de las restricciones establecidas por el propio entorno.

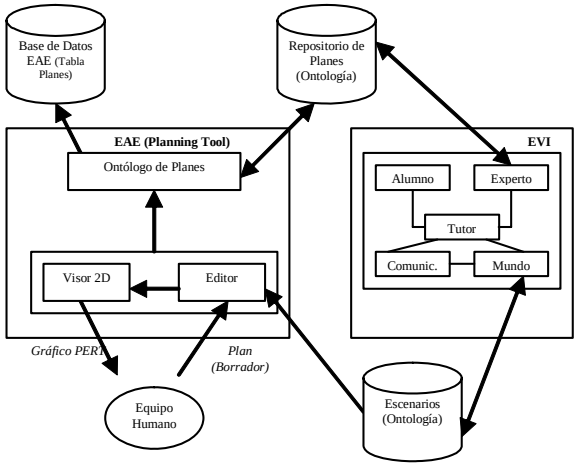


Figura 2: Arquitectura de la Herramienta de Planificación en el EAE

B. Visor 2D. Una vez elaborado el plan, el visor generará una vista gráfica —Grafico de PERT— del plan elaborado en el editor.

C. Ontólogo de planes. Genera, de acuerdo a una ontología predefinida, el plan de actuación diseñado por el equipo humano para la tarea en cuestión. Dicho plan mantiene un esquema similar al de los planes generados por el componente experto del EVI —a través de su agente planificador—. Los planes generados son almacenados en un repositorio de planes; de igual manera, cierta información que se requiere para la gestión de los planes por parte del EAE, es almacenada en una Base de Datos.

La figura 3 ilustra una vista del prototipo que se ha desarrollado para la herramienta de planificación.

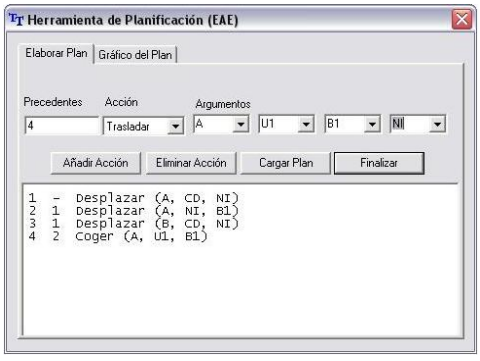


Figura 3: Vista de un Prototipo de la Herramienta de Planificación

4. Caracterización y Componentes de una Ontología

Una de las clasificaciones más conocidas en el ámbito de la Ingeniería Ontológica es la que Mizoguchi y cols. proponen en [5], éstos las clasifican en: ontologías de contenido (para la reutilización del conocimiento), ontologías de de comunicación (para compartir el conocimiento), de indexación (para la recuperación de conocimiento), y meta-ontologías (conocidas también como ontologías para representar conocimiento).

Con base en la clasificación antes descrita, una ontología de contenido puede definirse: como aquella que ofrece un vocabulario común para aquellos investigadores que necesitan compartir información en un dominio determinado, esta debe incluir definiciones de conceptos básicos en el dominio, así como de las relaciones que mantienen entre ellos. Las ontologías usualmente son organizadas en taxonomías y típicamente contienen primitivas de modelado como: clases, relaciones, funciones, axiomas e instancias [1]. Una breve descripción de estos componentes se ofrece a continuación:

Clase. Representa un concepto; puede ser todo aquello acerca del cual se dice algo.

Relación. Representa un tipo de asociación entre conceptos del dominio.

Función. Es un caso especial de una relación en el cuál el elemento enésimo de la relación es único para los n-1 elementos precedentes.

Axioma. Son utilizados para modelar sentencias que siempre son verdaderas.

Instancia. Representa a un elemento.

Las ontologías pueden ser modeladas utilizando diferentes técnicas de modelado del conocimiento (p.e. marcos y lógica de primer orden, técnicas de Ingeniería del Software, etc.) y pueden ser implementadas con varias clases de lenguajes (p.e. RDF, OWL, etc.). Así mismo, pueden ser expresadas mediante lenguaje natural (altamente informales), mediante alguna forma de lenguaje natural restringido y estructurado (semi-informales), mediante un lenguaje artificial formalmente definido (semi-formales), o mediante términos meticulosamente definidos y con una semántica formal (rigurosamente formales) [6].

4.1. Una Ontología para el Mundo Virtual

Para representar el conocimiento de todo aquello que se puede encontrar dentro de un escenario virtual, así como de aquello que se puede realizar en el mismo, se diseñó una ontología para el Mundo Virtual acorde con el tipo de escenarios considerados por el grupo investigador, para la realización de tareas de naturaleza socio-técnica en el EVIE.

La tabla 1 presenta los elementos considerados inicialmente para la ontología, así como sus definiciones y propiedades básicas.

Tabla 1: Elementos del Mundo Virtual

Concepto	Definición	Propiedades
Lugar	Ubicación identificable de un sitio en el espacio.	Nombre, coord. X, Y, Z).
Objeto-No-Manipulable	Elemento visible en el EV pero con el que no es posible interactuar.	Nombre_ON, Ubicación (lugar)
Objeto	Elemento del EV que puede ser utilizado para alguna función particular (herramienta, p.e. dosímetro), con la que es posible interactuar (manipular, p.e. un paquete radioactivo), o que puede ser utilizada para vestir (usar, p.e. casco).	Nombre_O, Funcionalidad.
Edificio	Construcción que se encuentra ubicada en algún lugar del espacio (p.e. Rectorado).	Nombre, Ubicación, Planta
Trayecto	Camino que interconecta a dos lugares.	Lugar1, Lugar2.
Personaje	Agente Humano o Inteligente que habita el EV.	Nombre, Rol.
Equipo	Personal encargado de alguna actividad.	Nombre, Personaje.
Vehículo	Medio de transporte utilizado por un personaje para desplazarse.	Medio (Terrestre, Marítimo, etc.).

Por otro lado, se analizaron las posibles acciones a realizar en el entorno virtual, identificando los parámetros para cada tipo de acción. La tabla 2 presenta las acciones consideradas inicialmente para el Mundo virtual.

Tabla 2: Acciones en el Mundo Virtual

Un parámetro:	Agacharse (Personaje), Levantarse (Personaje), Brincar (Personaje).
Dos parámetros:	Saltar_sobre (Personaje, Objeto), Usar (Personaje, Objeto), Conducir (Personaje, Vehiculo), Ponerse (Personaje, Objeto), Quitarse (Personaje, Objeto).
Tres parámetros:	Andar (Personaje, Inicio, Fin), Subir (Personaje, Objeto, Vehículo), Bajar (Personaje, Objeto, Vehículo), Reptar (Personaje, Lugar, Lugar), Nadar (Personaje, Lugar, Lugar), Bucear (Personaje, Lugar, Lugar), Coger (Personaje, Objeto, Lugar), Soltar (Personaje, Objeto, Lugar).

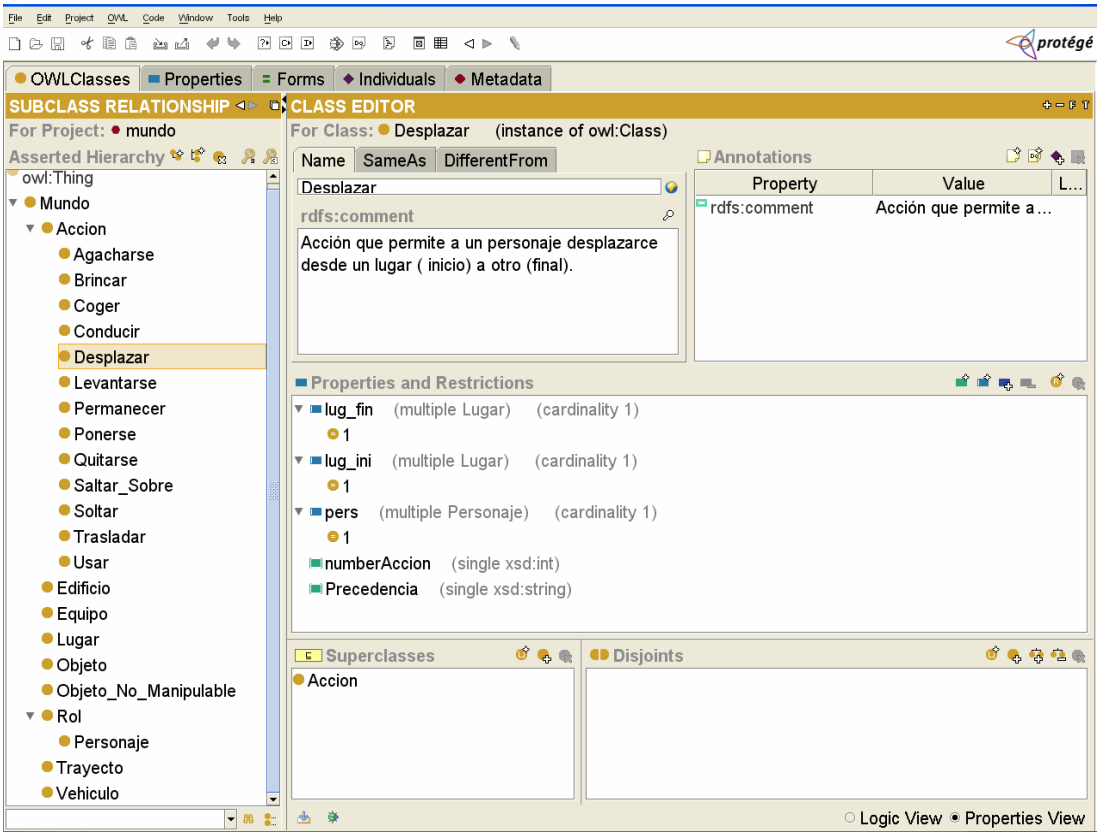


Figura 4: Vista de la Ontología del Mundo Virtual en Protégé.

Para la implementación de la Ontología del Mundo Virtual, se utilizó una aplicación de libre distribución denominada Protégé (http://protege.stanford.edu/); el entorno contiene un editor de ontologías que puede ser extendido con *plug-ins* para adicionarle mayor funcionalidad; en nuestro caso, se utilizó un *plug-in* que permite generar la ontología con base en la estructura del lenguaje OWL. La figura 4 ilustra una vista de la ontología del mundo virtual en el editor de Protégé.

4.2. Una Ontología para el Plan de Ejecución

Otro de los componentes en la arquitectura de la herramienta de planificación, es el módulo encargado de generar los planes diseñados por el grupo humano, éste módulo utiliza una Ontología que hemos definido para tal fin, a efectos de obtener planes en un formato similar a los planes generados por el componente experto del EVI. El EVI ha sido implementado mediante una arquitectura basada en agentes software [7] que actualmente utiliza para la generación automática de planes, el lenguaje JSHOP-2.

La Ontología definida para los planes presenta la siguiente estructura:

Definiciones:
{Var1, Var2, Var3, …, VarI}
Restricciones:
{R1, R2, R3, …, RJ}
Estado_Inicial:
{Var1=Edo1, Var2=Edo2, …, VarI=EdoI}
Estado_Final:
{Var1=EdoM, Var2=EdoN, …, VarI=EdoO}
Acciones:
{A1, A2, A3, …, AK}.

En donde mediante lógica de primer orden, se definen las variables en función de los conceptos, las restricciones en función de las relaciones y funciones definidas en la Ontología del Mundo Virtual, así como otras impuestas por el equipo humano para la realización de la tarea; así también el estado inicial define el estado que guardan los elementos en el escenario virtual al inicio de una tarea, el estado final define el estado deseado para los elementos en el escenario virtual al concluir la tarea, y las acciones representan las actividades simples (que solo requieren de un personaje para su ejecución) o compuestas (que requieren más de un personaje para su ejecución) listadas de manera secuencial, pero con indicadores de precedencia, los cuales permiten identificar el orden en que se podrán ejecutar dichas actividades.

La tabla 3 ilustra parte de un plan de ejecución que podría ser diseñado por el equipo humano utilizando el editor de la herramienta de planificación, para una tarea propuesta en un escenario virtual ubicado en la Facultad de Informática de la UPM. Para esta tarea se requerirían *instanciar* las acciones:

(D) Desplazar (Personaje, L-Inicio, L-Final).
(T) Trasladar (Personaje, Objeto, L-Inicio, L-Final)
(P) Permanecer (Personaje, Lugar)
(C) Coger (Personaje, Objeto, Lugar)
(S) Soltar (Personaje, Objeto, Lugar)

Así mismo se requerirían *instanciar* los conceptos siguientes:

Personajes: Agentes A, B, y C.
Lugares: Bloques B1, B2 y B4, Nodo de Inter. (NI), Central de destrucción (CD)
Objetos: Unidades radioactivas U1, U2, y U3.

Tabla 3: Parte de un posible Plan de Ejecución

No.	Precedente	Acción
1	-	D(A,CD,NI)
2	1	D(A,NI,B1)
3	1	D(B,CD,NI)
4	2	C(A,U1,B1)
5	4	T(A,U1,B1,NI)
6	3	D(B,NI,B4)
7	3	D(C,CD,NI)
8	5	S(A,U1,NI)
9	8	P(A,NI)
10	6	C(B,U3,B4)
11	10	T(B,U3,B4,NI)
12	7, 8	C(C,U1,NI)
13	12	T(C,U1,NI,CD)
14	11	S(B,U3,NI)
15	9, 14	C(A,U3,NI)

5. Trabajos en curso

En el ámbito de las ontologías, se continúa adicionando propiedades (restricciones) a la Ontología del Mundo Virtual, y se está refinando el esquema de representación que será utilizado en cada uno de los componentes —mediante lógica de primer orden— de la Ontología para el Plan de Ejecución, el cual será generado por uno de los componentes de la herramienta de planificación (Ontólogo de planes). Una vez concluidas ambas ontologías, se procederá a la integración de la herramienta de planificación al Entorno de Apoyo al Entrenamiento (EAE) para la realización de una prueba piloto con equipos humanos, a efecto de refinar los requisitos previamente identificados para el EAE en su uso durante la fase de Planificación.

Se continúa trabajando en el diseño del mecanismo que será utilizado para generar un primer modelo del grupo durante la dinámica colaborativa —co-construcción de un plan— en la fase de Planificación. Se ha determinado la utilización de oraciones de apertura [8] para monitorear las intenciones de comunicación de los integrantes del equipo durante su participación en la reunión virtual.

El equipo investigador tiene interés de continuar experimentando con técnicas de instrucción [9, 10] que pudiesen ser utilizadas en cada una de las fases de la estrategia de entrenamiento de equipos humanos propuesta conforme se vaya concluyendo el desarrollo del EAE y del EVIE.

Referencias

[1] Gruber, T. "A traslation approach to portable ontolgy specificatins", *Knowledge Adquisition*. 5 (2). 1993. pp. 199-220.

[2] Noy, N. & McGuinness, D.J. "Ontology Development 101: A Guide to Creating Your first Ontology", *Technical Report SMI-2001-0880*. University of Stanford. 2001.

[3] Aguilar, R.A. y De Antonio, A. "Entrenamiento de Equipos: Una estrategia asistida por entornos virtuales inteligentes", *IEComunicaciones. Revista Iberoamericana de Informática Educativa*. No. 2. 2005. pp. 25-33.

[4] Aguilar, R.A., Ramírez, J. & Calleja, J. "Evaluating a Collaborative Virtual Environments for Integration of Human Teams", *WSEAS Transactions on Advances in Engineering Education*. Issue 3, Vol. 2, 2005. pp. 235-242.

[5] Mizoguchi, R., Vanwelkenhuysen, J. & Ikeda, M. Task "Ontology for reuse of problem solving knowledge", In N. Mars (ed.) *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing (KBKS'95)*. University of Twente. IOS Press. 1995. pp. 46-57.

[6] Corcho, O. *A layered declarative approach to ontology translation with knowledge preservation*. Netherlands: IOS-Press. 2005. Chapter 1.

[7] De Antonio, A., Ramírez, J., Imbert, R., Méndez, G. & Aguilar, R. "A Software Architecture for Intelligent Virtual Environments Applied to Education", *Revista de la Facultad de Ingeniería, Universidad de Tarapacá*. Vol. 13 No. 1, Arica: Chile. 2005. pp. 47-55.

[8] McManus, M. & Aiken, R. "Monitoring computer-based problem solving", *Journal of Artificial Intelligence in Education*. 6 (4). 1995. pp. 307-336.

[9] Aguilar, R.A., de Antonio, A. & Prieto, M. "A procedure for the evaluation of instructional techniques used for the integration of teams", *Proceedings of IADIS International Conference on Cognition and Exploratory Learning in Digital Age*. Oporto: Portugal. 2005. pp. 459-462.

[10] Aguilar, R.A., de Antonio, A., Troncoso, B. & Imbert, R. "Scaffolding as Tutoring Approach in a Team Training Strategy", In L. Panizo, L. Sánchez, B. Fernández & M. Llamas (Eds). *Proceedings of 8th International Symposium on Computers in Education (SIIE 2006)*. León: Spain. 2006. pp. 352-360.

Experiencia en Team Software Process (TSP) y mejoras de Estimación, Calidad y Productividad de los Equipos en la Gestión del Software*

Bayona, S., Calvo Manzano, J., Cuevas, G., San Feliu, T.,
Dpto. Lenguajes y Sistemas Informáticos e Ingeniería del Software. Facultad de Informática
Universidad Politécnica de Madrid Campus de Montegancedo, 28660 Boadilla del Monte, Madrid, España.
E-Mail: sbayona@zipi.fi.upm.es;{jacalvo,gcuevas,tsanfe}@fi.upm.es

Resumen

En este artículo se describen los reales beneficios de adoptar procesos definidos y medidos como Team Software Process (TSP) y la evolución de equipos, los cuales muestran que a medida que los equipos evolucionan durante el ciclo de vida del proyecto y usan TSP, incrementan sus conocimientos y experiencia en su uso, éstos van mejorando en estimaciones de tamaño y esfuerzo, disminuyen la densidad de los defectos e incrementan la productividad. Los resultados se basan en un análisis exploratorio de datos del mismo proyecto software realizado por 22 equipos, dividido en dos ciclos con una duración total de 6 meses. Se recogió información de sus estimaciones de tamaño, esfuerzo, así como de sus valores reales. También se recogieron medidas de defectos. A partir de estos datos primarios, se ha realizado el análisis de las desviaciones obtenidas al estimar el tamaño a lo largo de los dos ciclos. Se ha analizado también la calidad en términos de defectos/KLOC, así como la planificación en términos de productividad y costes. Finalmente, se presentan las lecciones aprendidas de la utilización de la metodología TSP haciendo hincapié en la calidad del proceso de medida y en el entrenamiento.

1. Introducción

En los últimos años hemos asistido a un crecimiento exponencial de la demanda de software, que se ha venido aplicando en la resolución de tareas cada vez más complejas y proporcionando cada vez mayor valor añadido [8]. Los productos software siguen entregándose fuera de tiempo, exceden en coste y no cumplen con la calidad esperada por el cliente [9]. En estas circunstancias, se debe hacer la siguiente pregunta: ¿la industria del software está preparada para entregar el software que se necesita en los próximos años, con los niveles de productividad y calidad que se requieren?

Según Broadman [2] y Carreira [3] la respuesta es no, razón por la que en los últimos años se están desarrollando una serie de modelos y metodologías orientadas a minimizar la problemática a la que enfrenta la gestión de proyectos.

Las personas llevan a cabo cada una de las actividades y son quienes pueden llevar al éxito o al fracaso un proyecto. Un objetivo del Team Software Process (TSP) [5] [6] es construir y sostener un entorno de equipos que trabajen de manera cohesionada. Estos equipos deben atender las necesidades del negocio, mejorar la calidad, y reducir los costes y tiempos.

A pesar de que existen argumentos frecuentes de que la implementación de métodos rigurosos es muy costosa y conlleva mucho tiempo, se mostrará en este articulo como esto no sucede con los equipos que reciben entrenamiento TSP. En especial se tratará en este artículo, sobre el impacto positivo de este entrenamiento para alcanzar una mejor productividad y calidad del producto y reducir la densidad de los defectos.

Así mismo, con el dominio del TSP los equipos se encuentran más preparados para gestionar las necesidades de cambio del cliente, tienen la capacidad para evaluar inmediatamente como los cambios pueden afectar al plan detallado, basándose en la experiencia y las lecciones aprendidas.

* Este artículo esta patrocinado por las empresas ENDESA, DMR Consulting Foundation y SUN Microsystems a través de la "Cátedra de Mejora del Proceso Software en el Espacio Iberoamericano"

Esta experimentación está orientada hacia aspectos de calidad y productividad en la gestión de software con equipos que están trabajando alrededor de un objetivo común y donde cada persona asume un rol específico. Cada equipo está conformado por 5 ó 6 integrantes (Responsable de Desarrollo, Responsable de Equipo, Responsable de Planificación, Responsable de Proceso, Responsable de Calidad, Responsable de Soporte).

TSP está basado en los principios siguientes:

- Los ingenieros conocen más acerca de su trabajo y pueden hacer mejores planes.
- Cuando los ingenieros planifican su propio trabajo, ellos se comprometen con el plan.
- Para minimizar la duración del proyecto los ingenieros pueden balancear su carga de trabajo.
- Sólo la gente que hace el trabajo, puede recoger datos precisos.
- Para maximizar la productividad, hay que centrarse primero en la calidad.

Este artículo presenta una experiencia sobre la base del entrenamiento, uso del proceso TSP [4] [3] y análisis de los resultados que fueron obtenidos durante el desarrollo de un proyecto. Los datos históricos fueron proporcionados por 22 equipos del cuarto curso de la carrera de Ingeniería Informática de la Universidad Politécnica de Madrid a lo largo de la realización de un proyecto. Todos los equipos recibieron los mismos requisitos funcionales (todos debían de obtener el mismo producto).

El proceso seguido es el siguiente:

- La conformación de los equipos fue realizada de manera libre (se les permitió organizarse libremente entre ellos) y se les entrenó en técnicas de decisión, de factores humanos y definición de roles.
- En el Ciclo I, recibieron entrenamiento en TSP, se les entrega las especificaciones del cliente, se asignan los roles a cada miembro del equipo, se dan las instrucciones del proceso a seguir, los formularios a utilizar y se inicia el proyecto.
- En el Ciclo II, se refuerza el entrenamiento en TSP y se da entrenamiento en técnicas de revisiones e inspecciones.

Se parte de la hipótesis que los equipos al inicio carecían del conocimiento de TSP y que durante el segundo ciclo ya habían adquirido conocimiento de TSP y experiencia en el desarrollo del proyecto.

Los equipos tuvieron que registrar los datos en los formularios estándar TSP. Se registraron los valores estimados del tamaño y esfuerzo que tendría el producto en el primer y segundo ciclo de desarrollo.

La estimación es una de las actividades prioritarias para decidir cuanto tiempo se tardará en desarrollar el producto. Es importante estimar el tamaño basado en datos históricos, en este caso, solamente fue posible en el Ciclo II cuando ya contaban con los datos históricos del Ciclo I y con la experiencia acumulada, permitió a los equipos hacer una estimación más certera.

A lo largo del proyecto se recogieron los datos reales de tamaño y esfuerzo permitiendo así la comparación con los valores estimados. En términos de calidad se recogieron los datos de defectos eliminados.

El presente artículo está organizado de la siguiente manera: La sección 2 ilustra el análisis de la desviación de estimación de tamaño del producto, en la sección 3 se hace un análisis la densidad de defectos, en la sección 4 se ilustra el análisis de la productividad y en la sección 5 se analiza los costes. Para cada una de estas secciones se analiza la tendencia de los equipos y las mejoras obtenidas. Finalmente en la sección 6 se presentan las conclusiones junto con las lecciones aprendidas y trabajos futuros.

2. Error de estimación de tamaño

La hipótesis a probar en esta sección es: "*a través de los diferentes ciclos de desarrollo del proyecto la estimación de tamaño se ajusta gradualmente al tamaño real del producto a realizar*".

La Figura 1 muestra la distribución del Error de Estimación de Tamaño para el Ciclo I y el Ciclo II del proyecto, cuyos datos se han obtenido mediante la siguiente formula:

Error Estimación Tamaño =100 (Tamaño Real– Tamaño Estimado)/Tamaño Estimado*

2.1 Tendencia de los equipos

Como se observa en la Figura 1, los errores de estimación del Ciclo I son muy pronunciados porque algunos equipos sobreestimaron mientras que otros subestimaron el tamaño a realizar. Mientras que en el Ciclo II se observa que la curva es más estacionaria y cercana a cero es decir una mayor precisión en la estimación. Se observa como los equipos que sobreestimaron adquieren experiencia y mejoran considerablemente en la estimación de tamaño en el Ciclo II.

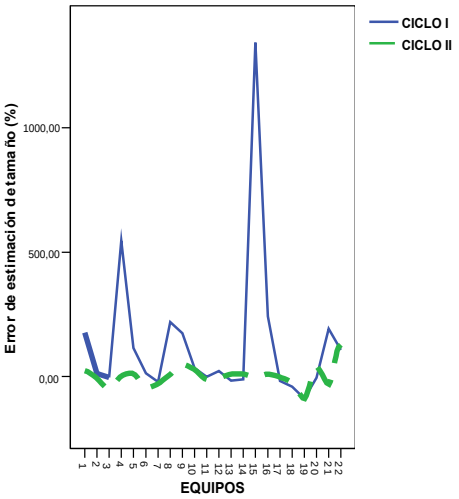


Figura 1. Error de Estimación de Tamaño para Ciclo I y Ciclo II

2.1 Análisis de mejoras en la estimación de tamaño

La Figura 2 fue generada con el paquete estadístico Statistical Product for Service Solutions (SPSS)[1] usando los datos de desviación de estimación de tamaño proporcionados por cada uno de los equipos. La caja central indica el rango en el que se concentra el 50% central de los datos, los extremos de la caja son el primer y tercer cuartil de la distribución. La línea central en la caja indica la mediana. Los extremos de los "bigotes" que salen de la caja son los valores que delimitan el 95% de los datos.

El eje vertical representa el error de estimación de tamaño y el eje horizontal los ciclos.

Los valores extremos representan el comportamiento anormal: en el Ciclo I dos equipos tuvieron un error de estimación superior al 500% (equipos 15 y 4). En el Ciclo II el valor extremo tuvo un error inferior al 200% (equipo 22).

Se utiliza el análisis de percentiles como una herramienta estadística para determinar la dispersión de los datos. El rango de percentiles del 25% al 75% del Ciclo I es del 192,56% de error en la estimación de tamaño, y este rango se reduce a 40,98% para el Ciclo II. El valor de la mediana en el Ciclo I es 17,75% y en el Ciclo II se reduce hasta 0,33%.

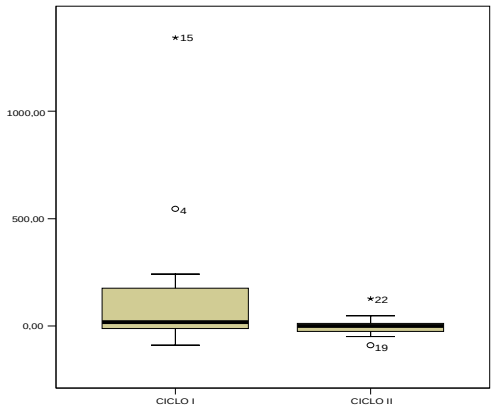


Figura 2. Error de Estimación de Tamaño

Esto es debido a que en el Ciclo II:

a) Los equipos ya cuentan con experiencia y conocimiento de técnicas de estimación, a diferencia del Ciclo I, en el que los equipos presentan dificultades de conseguir un grado aceptable de certeza mostrando subestimaciones y sobreestimaciones,
b) Cuentan con experiencia en el producto,
c) Cuentan con los datos históricos del Ciclo I.

De estos datos podemos deducir que a medida que los equipos van adquiriendo mayor experiencia mejoran la estimación a través de los ciclos.

Esta experiencia y los conocimientos adquiridos influirán en los demás aspectos como densidad de defectos, productividad y costes que se detallan a continuación.

3. Densidad de defectos

Para producir programas libres de defectos es importante medir la calidad de los programas, y con esta información dar los pasos para mejorar. Introducir defectos ocasionalmente en los programas es algo innato en los seres humanos. El reto es llegar a gestionarlos para reducirlos cada vez más alcanzando el valor lo más próximo a cero.

Un defecto es cualquier cosa que reduce la capacidad del software para cumplir de manera completa y eficiente la necesidad de los usuarios. El defecto es algo objetivo que se puede identificar, describir y contabilizar.

La construcción de software es un proceso al que se puede evaluar a lo largo de las etapas de los productos intermedios. Una alta densidad de defectos en los productos intermedios se traduce en un esfuerzo

añadido a causa del trabajo que hay que volver a realizar o el tiempo incurrido en solucionar defectos. Sin embargo, esta situación puede mejorar si se lleva un registro de los defectos que se están produciendo y si se identifican los factores que los ocasionan.

En esta sección se estudia la densidad de defectos producida por los equipos (número de defectos eliminados por cada 1000 líneas de código).

La hipótesis investigada en esta sección es: "*a medida que los equipos avanzan a través de los ciclos, el número de defectos inyectados, y por lo tanto eliminados, por cada 1000 líneas de código (KLOC) disminuye*".

La densidad de defectos es un indicador temprano de la calidad final y se define mediante la siguiente formula:

Densidad defectos=Defectos eliminados /KLOC

3.1 Tendencia de los equipos

La Figura 3 muestra la distribución de densidad de defectos para el Ciclo I y para el Ciclo II del proyecto con los datos de defectos inyectados y eliminados por cada 1000 líneas de código generados por los 22 equipos.

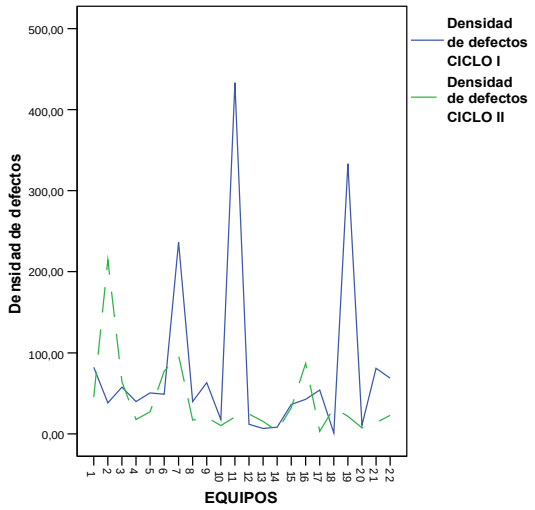


Figura 3. Densidad de defectos para Ciclo I y Ciclo II

En la Tabla 1 de los estadísticos descriptivos, se observa que la media de la densidad de defectos en el Ciclo I fue de 80 defectos/KLOC y en el Ciclo II esta densidad se reduce a 40 defectos/KLOC.

	Mín	Máx	Media	Desviación típica
Densidad defectos Ciclo I	1,0	433,3	80,0	110,3
Densidad defectos Ciclo II	2,9	214,7	40,0	47,2

Tabla 1. Estadísticos descriptivos de la densidad de defectos

En muchos ambientes de desarrollo, el registrar los defectos tiene sólo un objetivo de medir la calidad del código. Sin embargo se debe tener en consideración que los procesos de revisión e inspección, y las pruebas pueden también verse afectados por la experiencia y calidad de los revisores.

3.1 Análisis de mejoras en la densidad de defectos

La Figura 4 generada con el paquete estadístico SPSS muestra el diagrama de cajas para la densidad de defectos de los equipos. El eje vertical muestra la densidad de defectos y el horizontal los ciclos.

Los valores extremos representan el comportamiento anormal: en el Ciclo I tres equipos tuvieron 236 defectos/KLOC (equipo 7), 333 defectos/KLOC (equipo 19) y 433 defectos/KLOC (equipo 11) respectivamente. En el Ciclo II el valor extremo se reduce a 214 defectos/KLOC (equipo 2).

Del análisis de percentiles, se deduce que el rango del percentil 25% al 75% del Ciclo I es 55 defectos/KLOC y el rango obtenido en el Ciclo II es de 35 defectos/KLOC. El valor de la mediana en el Ciclo I es 46 defectos/KLOC y en el Ciclo II se reduce a 22 defectos/KLOC.

Así mismo analizando la desviación típica como una medida de dispersión, se observa que en el Ciclo I el valor es muy alto (110,3) mientras que en el Ciclo II baja a un valor cercano a la mitad (47,2).

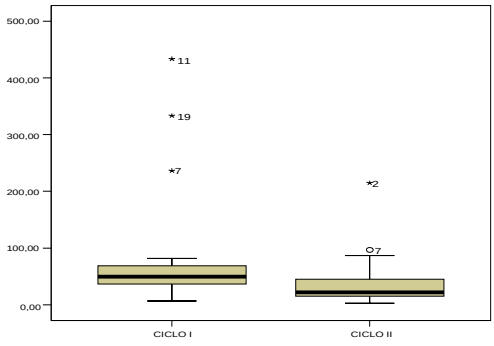


Figura 4. Reducción en la densidad de defectos

La variación del Ciclo I al Ciclo II fue debido a la introducción de técnicas de revisión e inspección.

De los datos anteriores podemos deducir que a medida que los equipos adquirieron mayor experiencia, así como la introducción de revisiones de diseño y código en el Ciclo II se produjo una disminución de los defectos inyectados, con lo que se obtuvo un producto de mayor calidad.

La reducción de la densidad de defectos total se traduce directamente en una reducción en la cantidad de trabajo que hay que rehacer y minimización de posibles retrasos y rechazo de parte del cliente. Un factor importante es que en el Ciclo II los miembros del equipo tienen la experiencia del Ciclo I y han aprendido de sus errores.

4. Productividad

Una métrica importante en toda organización es el índice de productividad, es decir, la generación de producto por unidad de tiempo.

La hipótesis a probar en esta sección es: "*a medida que se avanza en los ciclos de desarrollo del proyecto, la productividad se incrementa*".

La productividad está definida por la relación que existe entre la cantidad de líneas de código generadas por unidad de tiempo o esfuerzo realizado.

Productividad = LOC/Horas

4.1 Tendencia de los equipos

Para constatar la hipótesis planteada se tomaron los datos de 22 equipos.

La Figura 5 muestra la distribución de la productividad de los equipos, observándose una mejora en términos medios.

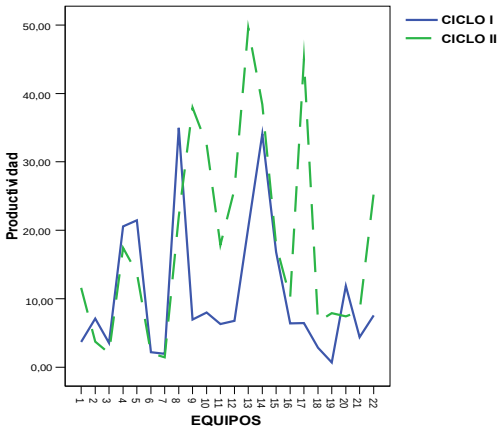


Figura 5. Productividad de los equipos por ciclo

Según los estadísticos descriptivos que se muestra en la Tabla 2, los valores medios se incrementaron desde 10,6 LOC/hora a 18,4 LOC/hora.

	Mín	Máx	Media	Desviación típica
Productividad CICLO I	0,6	35,0	10,6	9,8
Productividad CICLO II	1,4	49,0	18,4	14,5

Tabla 2. Estadísticos descriptivos de Productividad

4.2 Análisis de mejora en la productividad

En la Figura 6 se muestra un diagrama de cajas donde el nivel de productividad en el Ciclo II es cercano al doble del Ciclo I. En el Ciclo I se alcanzó un máximo de 35 LOC/hora mientras que en el Ciclo II se incrementa a 49 LOC/hora de productividad. El valor mínimo se incrementa de 0,6 LOC/hora a 1,4 LOC/hora.

El análisis de percentiles muestra que en el rango del 25%, en el Ciclo I se ha pasado de 3,6 LOC/hora a 7,2 LOC/hora en el Ciclo II. En el rango del 75% se ha pasado de 17,7 LOC/hora en el Ciclo I a 27,6 LOC/hora en el Ciclo II.

El valor de la mediana en el Ciclo I es 6,8 LOC/hora y en el Ciclo II aumenta a 15,6 LOC/hora.

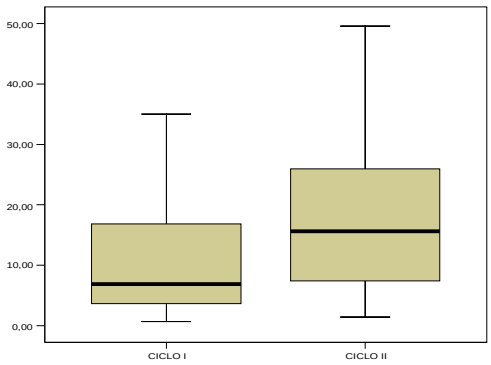


Figura 6. Mejora en la productividad

Este incremento se explica porque:

- En el Ciclo I el equipo de trabajo dedicará más horas de trabajo a la organización y planificación del proyecto, así como al conocimiento del dominio del problema a resolver.
- En el Ciclo I la productividad está condicionada al entrenamiento, los lenguajes de programación usados y la plataforma tecnológica.

- En el Ciclo II el equipo se va consolidando con una visión compartida para el logro de las metas y objetivos.

Con TSP se enfatiza en las fases tempranas del proyecto, dedicando más tiempo a los requerimientos, diseño de alto nivel, y revisando e inspeccionando el diseño. Así mismo el equipo se va consolidando con una visión compartida para el logro de las metas y objetivos.

5. Costes

La relación coste y productividad es otra inquietud en los proyectos software y se debe responder a la pregunta si están teniendo alta productividad pero a costes elevados.

Entonces, es importante medir si realmente el indicador de productividad alto permite reducir costes.

La hipótesis a probar es: "*a medida que los proyectos van mejorando a través de los ciclos, no sólo la productividad se incrementa, sino que se reducen significativamente los costes".*

En este caso se utiliza la relación que existe entre los costes del proyecto por líneas de código generadas y se define mediante la siguiente formula:

Productividad (Euros/LOC)=Coste proyecto/LOC

Coste proyecto= X Euros hora de trabajo*

5.1 Tendencia de los equipos

En la Figura 7 se observa una reducción de costes entre ciclos.

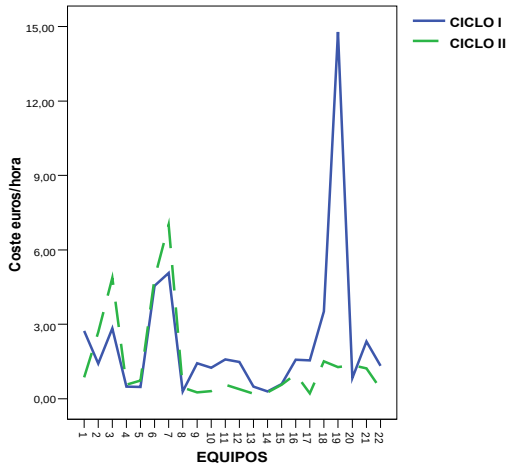


Figura 7. Comparativa de Coste por línea de código

En la Tabla 3 de los estadísticos descriptivos, se observa que la media de los costes relacionados con la productividad del Ciclo I es de 2,3 euros/LOC y para el Ciclo II es de 1,4 euros/LOC. Estas cifras indican que ha habido una reducción de costes y un crecimiento de la productividad.

	Mín	Máx	Media	Desviación típica
Costos en el Ciclo I	,2	14,7	2,3	3,0
Costos en el Ciclo II	,2	7,0	1,4	1,8

Tabla 3. Estadísticos descriptivos de los costes

5.2 Análisis de mejora en los costes

En el diagrama de cajas de la Figura 8 se observa como se reduce drásticamente la dispersión de los datos. Se han disminuido los máximos de 14,7 (equipo 19) a 7 euros/LOC (equipo 7).

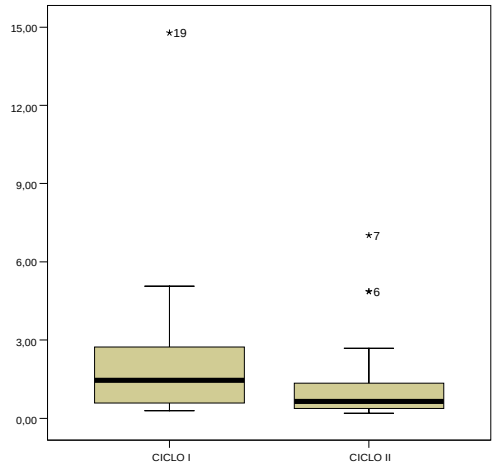


Figura 8. Comparativa de coste por línea de código

En el rango entre percentil del 25% al 75% del Ciclo I, el coste es de 2,2 Euros/LOC y, en este mismo rango, en el Ciclo II se reduce a 1 euro/LOC. El percentil 25% del Ciclo I es de 0,5 Euros/LOC y se reduce en el Ciclo II a 0,3 Euros/LOC.

El percentil 75% del Ciclo I es de 2,7 Euros/LOC y se reduce en el Ciclo II a 1,4 Euros/LOC. La mediana en el Ciclo I es 1,4 Euros/LOC y se reduce en el Ciclo II a 0,6 euros/LOC.

Esto se explica por que la experiencia del Ciclo I junto con el entrenamiento en TSP de los equipos ha hecho de que en el Ciclo II:

a) Hayan incrementado su productividad.
b) Se haya logrado reducir costes.

c) Las técnicas de pruebas mejoran y por lo tanto reducen tiempos sin descuidar la calidad.

Es decir a más entrenamiento se incrementa la productividad y se reducen costes.

6. Conclusiones y futuros trabajos

Las hipótesis planteadas en este experimento han sido comprobadas y en la Tabla 4 se presentan los resultados. Se observa que el error de estimación de tamaño disminuye, la densidad de defectos disminuye, la productividad se incrementa y los costes disminuyen con el uso y experiencia en TSP.

METRICA	Resultado	CICLO I			CICLO II		
		Med	Mín	Máx	Med	Mín	Máx
Error Estimación Tamaño (%)	-	97,8	-88,0	545,5	1,2	-89,2	127,6
Densidad defectos (defectos/KLOC)	-	80,0	1,0	433,3	40,0	2,9	214,7
Productividad (LOC/hora)	+	10,6	0,6	35,0	18,4	1,4	49,0
Costes (Euros/LOC)	-	2,3	,2	14,7	1,4	,2	7,0

Tabla 4. Cuadro comparativo entre Ciclo I yII

El rango de error de estimación de tamaño disminuye con el avance del proyecto. En el Ciclo I el rango se sitúa entre -88,0% y 545,5% con una media de 97,8%. En el Ciclo II el rango se sitúa entre -89,2% y 127,6% con una media de 1,2%.

La densidad de defectos disminuye con la introducción de técnicas de inspección y revisión. En el ciclo I, donde no se llevaron a cabo inspecciones ni revisiones, el rango se sitúa entre 1,0 defectos/KLOC y 433,3 defectos/KLOC. En el Ciclo II el rango se sitúa entre 2,9 defectos/KLOC y 214,7 defectos/KLOC con una media de 40 defectos/KLOC.

La productividad en el Ciclo II se incrementa porque los miembros del equipo han adquirido experiencia y conocimientos. En el ciclo I el rango se sitúa entre 0,6 LOC/hora y 35 LOC/hora. En el Ciclo II el rango se sitúa entre 1,4 LOC/hora y 49 LOC/hora con una media de 18,4 LOC/hora.

En consecuencia los costes bajan de una media de 2,3 Euros/LOC a 1,4 Euros/LOC.

Podemos concluir como lecciones aprendidas que:

- La aplicación de un método sistemático y disciplinado de gestión de proyecto software hace que sean posibles mejoras en estimación, densidad de defectos, productividad y, en consecuencia, reducción de costes.
- La estimación de software es una de las áreas con más oportunidades de mejora dada las dificultades de conseguir un grado aceptable de certeza durante las estimaciones.
- Es importante contar con un proceso de recopilación de datos y analizar métricas porque nos permite conocer cómo se evoluciona en el ciclo de producción, sí se está mejorando la calidad y productividad, y dónde conviene actuar para mejorar.
- El entrenamiento, habilidades y conocimientos necesarios para desarrollar las actividades encomendadas es un factor positivo en términos de productividad y calidad. Por tanto, invertir en entrenamiento del personal es fundamental.
- Los equipos al usar técnicas TSP reducen los defectos, incrementan la productividad y disminuyen los errores de estimación.
- Los equipos en TSP evolucionan desde una visión orientada al producto en el Ciclo I, hacia una visión orientada a la mejora del proceso en el Ciclo II.
- El introducir mejoras en los procesos que ya están definidos y medidos puede ser rápido y poco costoso como ha demostrado la introducción de inspecciones y revisiones.

Como futuros trabajos dentro de esta área se propone seguir estudiando:

- El comportamiento de cada uno de los miembros del equipo de acuerdo a los roles asumidos dentro del proyecto y su impacto en los resultados obtenidos.
- Los factores que influyen en la productividad tales como lenguajes y entornos de desarrollo empleados.

7. Referencias

[1] Alvarez Santos, Carmen (1994), Manual de SPSS: Estadística descriptiva.

[2] Brodman, J. & Johnson, D. “Project Planning: Disaster Insurance for Small Software Projects. LOGOS International, Inc. Proceedings from SEPG 2000: Ways to Make Better Software. 20-23 March 1999. Seattle, Washington.

[3] Carreira, M. & Román, I. “Estimación del Coste de la Calidad del Software a través de la Simulación del Proceso de Desarrollo”. Revista Colombiana de Computación, 2 (1) pp. 75-87. 2002.

[4] Donald R, McAndrews, “The Team Software ProcessSM (TSP):An Overview and Preliminary Results of Using Disciplined Practices”, November

2000, SEI Technical Report CMU/SEI-2000-TR-015, ESC-TR-2000-015.
[5] Humphrey,W. “TSP: Leading a Development Team”: Addison-Wesley Publishing Company, Inc., 1995.
[6] Humphrey, W. “Introduction to the Team Software Process”: Addison-Wesley Publishing Company, Inc., 1999.
[7] Noopur Davis, Julia Mullaney, “The Team Software ProcessSM (TSP) in Practice: A Summary of Recent Results”, September 2003, SEI Technical Report CMU/SEI-2003-TR-014.
[8] Pressman, R.S. Software Engineering: A Practitioner's Approach. 5th Edition - European Adaptation. McGraw Hill, 2004.
[9] Sommerville, I. and Sawyer, P. (1997), Requirements Engineering: A good practice guide. England: John Wiley & Sons.

Aprendizaje por Refuerzos en Problemas de Planeamiento con Restricciones

Ing. Pedro E. Colla[1] , Dr. Ernesto Martinez[2]
[(1)] *Centro de Desarrollo Córdoba – EDS ADU*
F.Alcorta 185 – P 5 -- EDS Argentina
X5000JHO - Córdoba - Argentina
pedro.colla@eds.com
pcolla@frsf.utn.edu.ar
[(2)] *Instituto de Desarrollo y Diseño – INGAR*
ecmarti@ceride.gov.ar
Avellaneda 3657 -- CIDISI – FRSF – UTN
S3002GJC - Santa Fe – Argentina

Abstract

Se plantea la aplicación de técnicas de Aprendizaje por Refuerzo (Reinforcement Learning) del campo de inteligencia artificial a la solución de problemas de planeamiento de proyectos con restricciones de recursos. Se repasa las referencias bibliográficas sobre el tema y se explora la utilización del Valor Presente Neto como función de utilidad con ventajas sobre alternativas basadas en la eficiencia en el uso de recursos asignados al proyecto por la organización. Se validan los resultados mediante una implementación de RL con el paquete piQLe y se comprueba las mejoras que la técnica propuesta implica.

Introducción

El planeamiento de proyectos bajo condiciones de restricción de recursos es un problema frecuentemente encontrado en varios dominios técnicos; en particular se lo encuentra en proyectos de mejora de procesos de software donde las organizaciones en marcha requieren la aplicación de recursos con conocimiento operativo pero al mismo tiempo son reticentes al asignarlos fuera de las actividades centrales de la organización. Dado que los problemas reales de organizaciones tienen una complejidad que hace que las soluciones de planeamiento se obtengan mediante heurísticas *ad-hoc* es natural preguntarse que tan bueno es el resultado de la aplicación de las mismas y es por lo tanto razonable esperar que la aplicación de técnicas de Aprendizaje por Refuerzo pueda contribuir a la obtención de soluciones óptimas a partir de la maximización de una determinada función de utilidad.

La bibliografía sobre el tema se concentra en funciones de utilidad que se basan en evaluar la eficiencia relativa en el uso de recursos respecto a una capacidad dada asignada al proyecto como máxima.

Como contribución original se explora en este trabajo la utilización del *Valor Presente Neto* como función de utilidad, si bien no es la elección realizada en las referencias bibliográficas su utilización se postula como incluso ventajosa en problemas de *Reinforcement Learning* aplicado a problemas de planeamiento al mismo tiempo que seguramente su interpretación resultará más familiar en procesos de decisión de negocios donde es habitual su uso. El tratamiento se mantiene deliberadamente en un marco simplificado que lo aleja de la complejidad de los problemas esperables en el mundo real, de alguna manera se asume que aplicando los recursos y tiempos adicionales apropiados cualquier problema de complejidad arbitraria podrá ser resuelto con las mismas técnicas. El trabajo se estructura comenzando con una descripción del dominio y el marco de aplicación (*Descripción del Dominio y Marco de Aplicación*) donde se delinea el dominio de la mejora de procesos de software como requiriendo planificaciones con restricción de recursos para a continuación hacer una breve reseña de las técnicas de Reinforcement Learning aplicadas en el trabajo. A continuación (*Experimento y Evaluación*) se exploran los detalles de implementación utilizando el framework piQLE y se discuten los resultados de las distintas corridas. Finalmente, se elabora brevemente sobre las limitaciones del enfoque utilizado (*Limitaciones*) y algunas conclusiones obtenidas durante la realización del trabajo (*Conclusiones*). El trabajo se completa con referencias bibliográficas.

Descripción del Dominio y Marco de Aplicación

Los esfuerzos de mejora de proceso de software (*Software Process Improvement* ó SPI) son llevados a cabo por las organizaciones como acciones ordenadas bajo el marco de proyectos; los mismos son estructurados como una red de actividades interdependientes que requieren la aplicación de recursos para su concreción.

Las técnicas concretas de planeamiento dependerán del dominio donde el plan será aplicado el que definirá los operadores de planeamiento admisibles, las reglas de estado-acción u otro formalismo lógico equivalente [GRANT05]. En el dominio de mejoras de procesos de creación de software las organizaciones utilizan modelos de referencia como el *Capability Maturity Model Integrated* (CMMI) patrocinado por el SEI en Carnegie-Mellon University para obtener guía sobre las actividades a ser realizadas y sus dependencias funcionales. La naturaleza de cada tarea determinara que duración tendrá para un nivel de esfuerzo (recursos) aplicado dado; la relación entre el esfuerzo aplicado y la duración no necesariamente es de naturaleza lineal; el aumento de recursos no llevará entonces necesariamente a reducir en forma proporcional la duración de la tarea sobre la cual se aplican. En este trabajo se asume que la optimización no incluye alterar la magnitud de los recursos asociados a una tarea como una forma de "modular" su duración y asi adaptarla a un marco de recursos restringidos.

Resultará natural apelar a la utilización de métodos como el denominado de *Camino Crítico (Critical Path Method – CPM)* para determinar el planeamiento óptimo en términos de calendario [CMU], en principio estos métodos asumen que las restricciones provienen de la duración de las tareas y sus dependencias de sincronización y paralelismo asumiendo la disponibilidad de recursos ilimitados.

Sin embargo, los proyectos de mejora de procesos de software rara vez se implementan a partir de contar con disponibilidad de recursos ilimitados; en general los mismos requieren recursos claves de la organización los que al asignarse sobre operaciones "en marcha" solo se podrán utilizar en magnitudes muy reducidas [COLLA05].

En este escenario la verdadera naturaleza del plan resultará dictada por la capacidad de obtener los resultados finales en la forma más temprana posible sin exceder los recursos que la organización define como disponibles.

El problema de resolver un plan óptimo con restricciones de recursos se considera como *NP-Complete*[1] lo que reduce las esperanzas de obtener soluciones globalmente óptimas.

[1] Wikipedia (http://en.wikipedia.org/wiki/NP-complete)

Se tomará como punto de partida que la optimización de un plan de estas características comienza con la disponibilidad de un plan completamente definido y optimizado con técnicas de camino crítico bajo la hipótesis de recursos ilimitados. Sobre este se empieza a aplicar heurísticas *ad-hoc* haciendo que las tareas se desplacen en el tiempo de forma que no se excedan los recursos disponibles utilizando como guía la maximización de la función de utilidad seleccionada.

Planeamiento

Dada una red de actividades con sus dependencias funcionales previamente optimizada por el método del *Camino Crítico* la organización define que nivel de recursos máximos desea abocar al proyecto; computado el perfil de recursos se pueden identificar los tramos del plan inicial donde se requieren más recursos que los disponibles. El problema de la aplicación de técnicas de aprendizaje reforzado a la solución de problemas de *Job Shop Scheduling* ha sido abordado en sucesivos trabajos por Zhang & Dietterich [ZHANG95] quienes se concentran en proyectos que deben desenvolverse en contextos de restricción de recursos y que pudieran necesitar recursos de distinto tipo, cada uno provisto desde una fuente ó "pool" con su capacidad máxima.

Si bien se puede asimilar esta característica a proyectos de mejora de proceso en función de considerar grupos de conocimiento (*skill*) que necesitan trabajar cooperativamente en las distintas actividades del plan o condiciones de dispersión geográfica (sitios) actuando coordinadamente en partes del plan se tomará esta posibilidad como un mero incremento de complejidad del problema que no afecta significativamente los resultados.

Por lo tanto se abordarán los resultados de acomodar los excesos en los recursos requeridos mediante el desplazamiento temporal de las actividades que requieren de esos recursos. En la figura [*Figura 1*] se muestra como una situación de excedente en uso de recursos en un plan dado puede solucionarse mediante la extensión de la duración total.

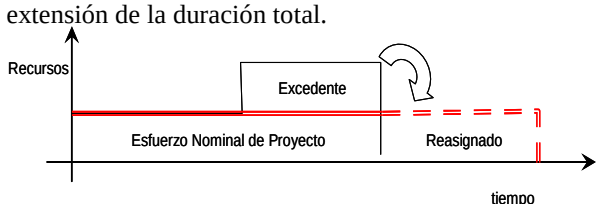


Figura 1 Nivelación de Plan por Restricciones

Por lo tanto se considerará el problema de la capacidad de recursos desde el punto de vista que el

plan deberá ejecutarse sin involucrar más staff que al asignado, proviniendo ese staff de una única locación geográfica (*pool único*) y siendo similares sus competencias (*skill*).

Identificadas las tareas en éstas condiciones estas deben ser movidas en el tiempo a momentos donde exista una disponibilidad de recursos que pueda ser aplicado. Si durante la duración del proyecto tal como fue obtenida por el método de camino crítico no existe excedente de recursos que permita satisfacer esta condición entonces la finalización del proyecto se extiende de forma de hacer uso de los recursos disponibles por más tiempo.

En términos prácticos se le aplica un desplazamiento *Δt (shift)* al tiempo más temprano de ejecución de forma que no empiece hasta el momento en que existan recursos disponibles.

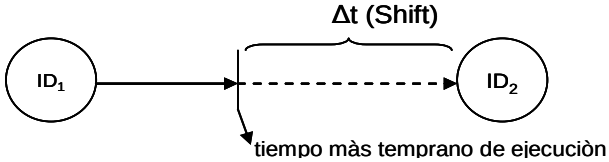


Figura 2 Desplazamiento de Tareas por Restricción

Si la tarea desplazada pertenece al "*camino crítico*" cualquier corrimiento del momento más temprano de ejecución producirá la extensión del proyecto completo, mientras que si la actividad no forma parte del camino crítico este desplazamiento si es menor que su *margen libre*[2] no impacta en la extensión total del proyecto.

Naturalmente cualquier heurística de solución de este problema tendrá debidamente en cuenta como primer paso de optimización el intentar solucionar los excedentes de recursos primero mediante movimientos de tareas no pertenecientes al camino crítico y dentro de su margen libre[3].

Para tener en cuenta algorítmicamente el acomodamiento de tareas para ajustar recursos se desarrolla una modificación del cómputo de camino crítico donde la *fecha de comienzo temprana (t_{se}), fecha de terminación temprana (t_{ee}) , fecha de comienzo tardía (t_{ls}) y fecha de terminación tardía (t_{ls})* según las siguientes relaciones:

$$t_{se}(n) = \max\{t_{ee}(n-1)\} + shift(n)$$
$$t_{ee}(n) = t_{se}(n) + duration(n)$$
$$t_{le}(n) = \min\{t_{ls}(n+1)\} - shift(n+1)$$
$$t_{ls}(n) = t_{le}(n) - duration(n)$$

Ecuación 1

En el enfoque propuesto por Zhang & Dietterich [ZHANG95,00,01,01B] se apela a utilizar coeficientes basados en la mejora en la utilización de recursos respecto al plan de partida.

Con éste enfoque se comienza por definir una *Función de Utilización de Recursos RUI(i.t) (Resources Utilization Index)* para el recurso de tipo i en el tiempo t tal que

$$RUI(i,t) = \max\left\{1, \frac{u(i,t)}{capacity(i)}\right\}$$

Ecuación 2

Donde *capacity(i)* es la capacidad total, supuesta fija durante la duración del plan, del i-esimo recurso (supuestos múltiples recursos concurrentes) y u(i,t) la utilización del mismo i-esimo recurso en el periodo t.

Si el recurso no está sobre utilizado RUI es 1, caso contrario es mayor que uno en proporción a su sobre uso.

El índice de utilización total para un plan de duración l es la suma de todos los índices de utilización para todos los recursos involucrados.

$$TRUI = \sum_{i=1}^{n}\sum_{t=1}^{l} RUI(i,t)$$

Ecuación 3

Dadas estas definiciones se introduce el concepto de *Coeficiente de Dilación de Recursos (Resource Dilation Factor RDF).*

$$RDF(s, s_0) = \frac{TRUI(s)}{TRUI(s_0)}$$

Ecuación 4

Zhang & Dietterich proponen utilizar la negativa del factor RDF como función de refuerzo.

Este factor identifica los desbordes de recursos respecto a la capacidad de los mismos pero no captura componentes de premio por *tiempo* y por *riesgo* típicamente presentes en inversiones bajo contextos de negocios.

Valor Presente Neto como Refuerzo

Se analiza en este trabajo la posibilidad de utilizar en cambio el *Valor Presente Neto* que si captura ambos premios.

[3] Tiempo que una tarea puede demorar su comienzo o su fin antes de tornarse parte del camino crítico.

El proyecto devengará durante su ejecución *Costos (C)* los cuales típicamente serán proporcionales al esfuerzo aplicado al proyecto y obtendrá al finalizar *Beneficios (I)*, para estudiar el impacto de ambos puede recurrirse al *Valor Presente Neto (VPN)* de los flujos descontados a un *costo de oportunidad (r)* que resulte relevante al riesgo del capital de la organización, esta medida es a menudo referenciada por algunos autores como la más potente para la toma de decisiones de inversión [BREALEY94].

$$VPN(t) = \sum_{t=0}^{t_{proyecto}} \frac{(I_t - C_t)}{(1+r)^t}$$

Ecuación 5

Si, como resultado del re-acomodamiento de actividades la duración total del proyecto resulta extendida el beneficio del mismo (supuesto devengado con la terminación del plan) se mueve hacia el futuro por lo que disminuye la contribución al *Valor Presente Neto* de la totalidad del esfuerzo reduciendo el valor del proyecto para la organización.

Se visualiza entonces la solución óptima del problema como el equilibrio entre que los recursos aplicados no superen el máximo definido a nivel organizacional y la duración total del plan, el óptimo se logrará cuando el plan se extienda lo mínimo posible al mismo tiempo que no supere los recursos máximos que se le asignaron.

Reinforcement Learning

En *Aprendizaje por Refuerzo (Reinforcement Learning , RL)* se enmarcan un conjunto de técnicas de aprendizaje donde el agente aprende ejecutando acciones por las cuales recibe castigos o recompensas, también llamados refuerzos.

Un sistema de aprendizaje por refuerzo consiste en un entorno, un agente y una función de refuerzo [SUTTON03 ,NILSSON96 , ALPAYDIN0].

El entorno contiene:

- S un conjunto finito de estados s
- A(s) s ϵ S los conjuntos finitos de acciones disponibles en cada estado.
- P(s'|a,s) la función de probabilidad de transición.

Los refuerzos vienen dados por R(s',a,s), la función de refuerzos. La función de probabilidad de transición P(s',a,s) es la probabilidad de ir a un estado s' dado que el agente se encuentra en el estado s y ejecuta la acción a. La función de refuerzo se especifica para un determinado comportamiento en un determinado problema.

Para definir cual es la acción a ser aplicada en un determinado momento recurrimos a la formulación de políticas y para comparar distintas políticas usamos la Función de Valor V(π)

$$V(\pi) = \frac{1}{|S|} \sum_{s \in S} V^{\pi}(s)$$

Ecuación 6

Los métodos de Reinforcement Learning aprenden una política para seleccionar acciones en un espacio de problemas dado. La politica dicta, en cada estado (s), cual es la acción (a) que se realizará. Cuando la acción es elegida y aplicada el estado cambia a un *nuevo estado (s')* en el espacio de problema y el sistema en su conjunto recibe un refuerzo dado por la *función de refuerzo R(s',a,s).*

Para poder ver el problema de planeamiento desde una óptica de Reinforcement Learning (RL) se debe describir el espacio del problema (ambiente) y la función de refuerzo.

Se sigue la propuesta realizada por Zhang [ZHANG01B] de utilizar como estados los distintos planes obtenidos a partir de la aplicación de acciones. El plan de comienzo (s_0) es el plan de camino crítico con recursos ilimitados discutido en la sección anterior. Las acciones (a) consistirán en la aplicación de *desplazamientos o shifts* Δt (positivos o negativos) en los comienzos tempranos de las tareas, cada vez que se aplique un desplazamiento de este tipo a una tarea se pasará el plan del estado s a s'.

La función de refuerzo R(s,a,s') deberá proveer un refuerzo negativo (penalización) toda vez que luego de la aplicación de una acción el plan siga teniendo excedidos los niveles de los recursos permitidos; se considerará al estado como final toda vez que el plan en su conjunto no exceda los niveles de recursos permitidos en ningún momento.

La recompensa (*reward*) es propuesta por Zhang como

$$R(s_{n-1}, a, s_n) = -RDF$$

Ecuación 7

Como alternativa en este trabajo se propone el Valor Presente Neto de los flujos del proyecto con el mismo propósito; cuanto menos duración tenga el plan más cercanos a su comienzo ocurrirán los beneficios y por lo tanto el valor presente de los mismos influenciará en mayor medida el valor total del proyecto.

$$R(s_{n-1}, a, s_n) = \sum \frac{(V_i - C_i)}{(1+r)^i}$$

Ecuación 8

Cuando un determinado estado continue utilizando más recursos que los disponibles se aplicará un refuerzo ("castigo"):

$$R(s_{n-1}, a, s_n) = -0.01$$

Ecuación 9

Esto producirá que cuantos menos pasos infructuosos se requiera para alcanzar una solución mayor será el valor de la recompensa acumulado. El algoritmo de aprendizaje deberá permitir obtener una *política óptima* (π) que permita elegir la siguiente acción

$$a = \pi(s)$$

Ecuación 10

Definiendo una función asociada, llamada *función de valor* $V_\pi(s)$ que pronostique el valor acumulado de las recompensas que se obtendrán si se sigue la política π desde el estado s en adelante.

En nuestro problema

$$V_\pi(s) = \sum_{j=0}^{N} R(s_{j+1})$$

Ecuación 11

Donde N es el número de acciones necesarias para alcanzar un calendario libre de conflictos.

El problema central de esta aplicación de Reinforcement Learning consiste en encontrar la función de valor de la política óptima

$$V_\pi^*(s) = V_{\pi^*}$$

Ecuación 12

Una vez que se aprendió la función de valor óptima esta se puede transformar en la política óptima mediante la aplicación de una búsqueda para cada estado de cual es la acción que maximizará el valor de esta función.

Para aprender el valor de esta función se pueden utilizar varios métodos; en este trabajo se utilizarán los algoritmos denominados Q-Learning., los algoritmos en forma detallada son expuestos en Sutton & Barto [SUTTON03] y no serán repetidos aquí excepto en forma muy general.

Q-Learning

El algoritmo conocido como Q-Learning fué desarrollado por Watkins como un algoritmo de Time Difference de un paso cuya expresión de la *función de valor de estado-acción Q(s,a)* es:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \gamma \max Q(s_{t+1}, a\ ) - Q(s_t, a_t)\right]$$

Ecuación 13

En este caso la función acción-valor aprendida aproxima directamente la óptima independientemente de la política que se persiga. Esto simplifica notoriamente el análisis del algoritmo y las condiciones de convergencia.

Experimento y Evaluación

A los efectos de realizar el análisis propuesto se postula una red de tareas simple, cualquier red servirá con éste propósito pero para mantener cierta consistencia respecto del uso de las técnicas bajo estudio al campo de mejora de procesos de software se utilizará una versión simplificada de plan compuesto por las *Areas de Proceso de CMMI Nivel 2*. En proyectos reales de mejora de procesos seguramente las dependencias reales se establecerán como la relación entre las distintas *Prácticas Genéricas y Específicas* de las *Areas de Proceso*, pero para permitir avanzar con el problema se asume un conjunto de dependencias a nivel de las *Process Areas (PA)* directamente. Se procura evitar complicar innecesariamente el análisis utilizando la red de prácticas genéricas y específicas puesto que esto redundaría en tiempos de procesamiento mucho más prolongados sin agregar elementos importantes al análisis preliminar como el abordado en éste trabajo.

La red utilizada se muestra en la figura [Figure 3] .

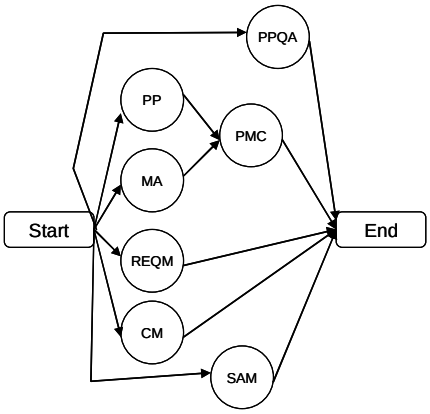


Figura 3 Red de Actividades (Ejemplo)

Esta elección es por supuesto arbitraria y cualquier otra red de complejidad equivalente y parámetros de planeamiento razonables podrá ser utilizada en su lugar, incluso la dependencia entre distintas Areas de Proceso es elegida en forma arbitraria puesto que seguramente habrá muchas otras alternativas igualmente viables.

En este dominio en particular se simplifica sin perder generalidad al considerar que cada actividad requiere un esfuerzo del orden de 6 Staff/Mes y durará

6 meses es aproximadamente consistente con los valores promedios referenciados por la bibliografía para estas iniciativas [COLLA05].

Puede considerarse apropiado para una organización el asignar en el entorno de un 10% de sus recursos totales al esfuerzo de mejora de proceso; por lo tanto para una organización cuyo tamaño total sea del orden de 20 a 40 personas el alocar entre 2 y 4 personas (equivalente) será probablemente una magnitud de recursos asignados realista.

El completar la implementación (asumido que se obtiene la evaluación CMMI Nivel 2) redundará en una mejora de la productividad en el orden del 10%, proyectando el valor presente de esa mejora vista como un flujo continuo a futuro resulta en el equivalente a unas 200 Staff/Mes. Estos valores son un artificio conveniente para expresar costos y beneficios en términos de esfuerzo con magnitudes aproximadamente consistentes con las experimentadas por organizaciones reales.

Implementación

Para implementar este problema se recurre a la utilización del paquete *piQLe* [DECOMITE05], para hacer uso del mismo será necesario definir el problema en términos de la implementación de una serie de clases Java y la configuración de los recursos del framework necesarios para realizar las corridas.

Se empieza por implementar una representación para las actividades y para el conjunto de ellas estructurada en la forma de una red de dependencias para formar un plan. Finalmente, estas clases están integradas en una clase representativa del problema que las instancia y contiene.

Desde el punto de vista del ambiente piQLe se implementa una clase para las acciones; cada acción estará representada por el código de tarea y el desplazamiento de tiempo (+ ó -) que se aplica.

El estado es representado por un objeto del problema que esencialmente instancia una red de actividades con un dado estado de relaciones entre actividades.

El ambiente del problema se implementa a partir de una clase la cual es una extensión de la clase *Contraintes* del framework y es donde se implementan las condiciones de inicialización, la lista de acciones aplicables, el cálculo del estado siguiente cuando una determinada acción es aplicada, el cálculo de la recompensa y el cálculo de la condición de finalización (además de otros métodos de carácter utilitario y funcionales a la operación del framework mismo).

Finalmente se implementa una clase de alto nivel cuyo propósito es instanciar las clases piQLE y realizar la ejecución de las corridas de aprendizaje propiamente dichas; en la misma se selecciona el algoritmo a ser utilizado por la corrida asi como su parametrización respectiva, en este caso se utilizó el algoritmo SelectionneurQL para Q-Learning. Los parámetros utilizados con las diferentes corridas fueron:

Parámetro	Valor utilizado
ε	0.5 (inicial)
α	0.3

Figura 4 Parametros de Corridas

Corridas

Se despliegan a continuación los resultados de las diferentes corridas realizadas. Lo que interesa en realidad es capturar la tendencia de convergencia a resultados óptimos y las bondades relativas del algoritmo Q-Learning usando como función de utilidad el *Resources Dilation Factor (RDF)* ó el *Valor Presente Neto (VPN)* más que la obtención de una solución óptima. Por lo tanto se adoptó el criterio de mantener el número total de iteraciones en 1000 y el número máximo de movimientos permitidos en 500; ambas medidas limitan cuan óptimo es el resultado final pero mantiene los tiempos de corrida en los distintos escenarios dentro de duraciones razonables. Se registra el mejor resultado obtenido dentro del horizonte de corridas.

Se restringe el nivel de recursos que la organización puede aplicar a este plan a 3 Staff, esto garantiza para los parámetros de esfuerzo de la red y dependencias que el plan operará en condiciones de restricción de recursos.

Se corren sucesiones de episodios donde se utiliza al *Resource Dilation Factor (RDF)* como función de utilidad y otra sucesión de episodios donde se utiliza al *Valor Presente Neto (VPN)* como función de utilidad.

Independientemente de cual es utilizado como función de utilidad se calculan ambos como para poder comparar la evolución en uno u otro caso.

Se puede observar que cuando se utiliza VPN (linea sólida) como función de utilidad se obtiene una convergencia más veloz a valores óptimos de RDF (cercanos a 1) que cuando se utiliza al RDF (linea discontinua) mismo como función de utilidad [*Figura 5 Comparación de Evolución de Resource Dilation F*]

Al mismo tiempo la evolución de soluciones con mejor Valor Presente Neto y en menor tiempo se produce cuando se utiliza al VPN como función de utilidad [*Figura 6 Comparación de Evolución de Valor Presente N*]

Estos resultados sugieren una mejor performance del Valor Presente Neto como función de utilidad en éste tipo de problemas.

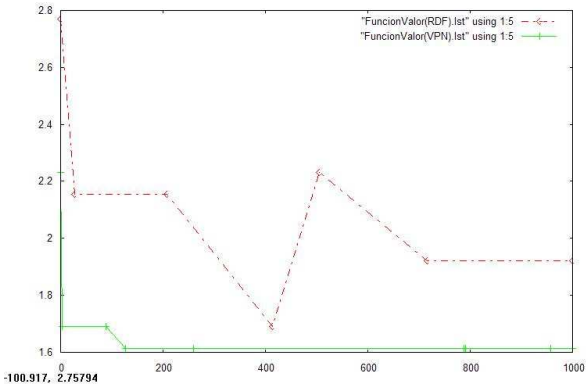


Figura 5 Comparación de Evolución de Resource Dilation Factor en función del número de episodios (linea sólida es VPN como función de utilidad y linea discontinua es RDF como función de utilidad)

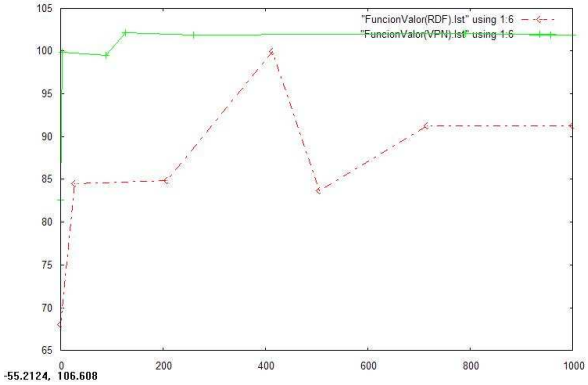


Figura 6 Comparación de Evolución de Valor Presente Neto en función del número de episodios (linea sólida es VPN como función de utilidad y linea discontinua es RDF como función de utilidad).

Limitaciones

Las limitaciones del presente análisis derivan de utilizar un problema expresado en forma simplificada comparado con la aplicación de la técnica a problemas reales (complejos) de planeamiento.

La bibliografía plantea el uso de heurísticas complejas para optimizar el proceso de búsqueda de la solución óptima con recursos computacionales razonables; estas heurísticas son dejadas de lado y por lo tanto se sacrifica que tan bueno es el resultado final en función de investigar los postulados bajo evaluación.

Adicionalmente, el análisis de un problema de mundo real no puede excluir del proceso de optimización la variación del esfuerzo aplicado a las actividades como un posible motor de satisfacción de restricciones de recursos; la simplificación operada en éste trabajo es que la duración de las tareas es relativamente inelástica a la variación del esfuerzo alocado a la misma y por lo tanto si una actividad no tiene recursos suficiente la única forma de solución es corriendo la actividad misma en el tiempo (en lugar de ejecutarla con recursos menores por más tiempo).

Existe por supuesto la incognita sobre si las técnicas escalarán apropiadamente entre las magnitudes de ambos extremos de uso aunque la elaboración de ese interrogante se deja fuera del alcance de éste trabajo.

Conclusiones

Es evidente que las técnicas de Reinforcement Learning pueden ser exitosamente aplicadas al problema de planeamiento con recursos restringidos. Esto no debería ser una sorpresa porque las referencias bibliográficas ya reportan la aplicación exitosa de esta técnica en este dominio; sin embargo este trabajo extiende las conclusiones a la utilización de otras funciones de utilidad como el Valor Presente Neto que resulta más convencional para medir el valor de un proyecto en un contexto de proyectos de negocios, al menos en el dominio propuesto.

Se interpreta que los resultados obtenidos (que son menores que los óptimos) resultan de un tema de escala y potencia de procesamiento más que de algoritmo propiamente dicho.

La obtención de soluciones óptimas invita a la utilización de éste enfoque tanto para obtener un plan como también como herramienta de validación (control de calidad) de planes realizados con otras heurísticas en forma manual o automática.

La aplicación de Reinforcement Learning en este dominio sufre claramente como en tantos otros dominios de la "maldición de la dimensionalidad", el número de estados posibles y acciones posibles es tan elevado (aún en un problema mínimo como el utilizado de ejemplo) que la potencia de procesamiento a poner en juego es considerable; es importante explorar otros algoritmos que tengan en cuenta específicamente este problema tal como propone Paganelli [PAGANELLI06]. ó técnicas de Relational Reinforcement Learning, lo que será foco de trabajos posteriores.

Referencias

[Alpaydin04] Introduction to Machine Learning, Alpaydin E. The MIT Press, 200

[Brealey94] Brealey R. & Myers S. "Fundamentos de Financiacion Empresarial" , McGraw-Hill, 4th Edition 1994.

[Colla05], Un contexto para la formulación de modelos sistémicos Colla P: Montagna M.. ASSE-JAIIO 2005.
[Colla06] Marco extendido para la evaluación de iniciativas de mejora en procesos en Ingeniería de Software Colla P.:, JIISIC 2006 (Puebla, México).
[CMU00] Fundamental Scheduling Procedures – Project Management for Construction (Chapter 10) – Carnegie.Mellon University.
[DeComite05] A Java Platform for Reinforcement Learning Experiments DeComité F. PDMIA 2005
[Grant05] Generating Plans from a Domain Model-Grant T.J. 14th UK Planning and Scheduling SIG workshop University of Essex, Colchester 22-23 Nov 95
[Markovitch03] Speedup Learning for Repair based Search by Identifying Redundant Steps, Markovitch S. Shatil A. Journal of Machine Learning Research (2003) p.649-682
[Nilsson96] Introduction to Machine Learning, Nilsson Nils J. 1997.
[Paganelli06] Una Técnica de Aprendizaje por Refuerzo para la Resolución de Problemas Complejos Paganelli, F. Otamendi F. Matt A.D. 8vo Simposio Argentino de Inteligencia Artificial (ASAI 2006).
[Rabideau03] Using Iterative Repair to Automate Planning and Scheduling of Shuttle Payload Operations, Rabideau G .,Chien S., Willis, J. Mann.T.American Association of Artificial Intelligence (www.aaai.org)
[SEI]CMMI, Capability Maturity Model Integration (2002), Version 1.1 CMMISM for Software Engineering (CMMI-SW, V. 1.1) Staged Representation CMU/SEI-2002-TR-029 ESC-TR-2002-029 CMMI Product Team, SEI Carnegie-Mellon University.
[Sutton03] einforcement Learning: An Introduction Sutton R.S and Barto A.G Bradford Book The MIT Press Cambridge, Massachusetts
[Zhang95] A Reinforcement Learning Approach to Job-Shop Scheduling Zhang, W. Dietterich, T.G. IJCAI95.
[Zhang00] Solving Combinatorial Optimization Tasks by Reinforcement Learning: A general methodology applied to Resource-Constrained Scheduling. Zhang W. Dietterich T. Journal of Artificial Intelligence Research (2000) 1-1
[Zhang01] High-Performance Job-Shop Scheduling with Time-Delay TD(Lambda) Network, Zhang W. Dietterich T.G.
[Zhang01B] Value Function Approximations and Job-Shop Scheduling, Zhang W. , Dietterich T.G.

TUTORIALES

Tutorial: Uso de Esquemas Preconceptuales para la generación automática de Diagramas de Clases, Comunicación y Máquina de Estados[1]

Carlos Mario Zapata J.
Grupo de Ingeniería de Software, Escuela de Sistemas, Universidad Nacional de Colombia
cmzapata@unal.edu.co

1. Objetivos

- Suministrar elementos básicos de modelamiento de software orientado a objetos, incluyendo el desarrollo de tres esquemas conceptuales de UML: los diagramas de clases, máquina de estados y comunicación [1, 2]. Se emplea para ello una representación intermedia denominada "Esquema Preconceptual" [3] y un conjunto de reglas heurísticas de transformación.
- Comprobar en la práctica la utilidad de los Esquemas Preconceptuales y su forma de obtención a partir de un lenguaje controlado.

2. Metas

- Presentar las características mínimas de los diagramas de clases, comunicación y máquina de estados, pertenecientes al estándar UML versión 2.0 [1].
- Estudiar y practicar la sintaxis de los Esquemas Preconceptuales [3].
- Realizar ejercicios prácticos de transformación de Esquemas Preconceptuales en los tres tipos de diagramas de UML.
- Discutir la manera de incorporar los Esquemas Preconceptuales en los métodos de desarrollo de software orientado a objetos. Un ejemplo de ello se puede apreciar en [4].
- Presentar el prototipo de herramienta CASE denominado UNC-Diagramador, un tipo de herramienta enfocado en la fase de definición de una pieza de software, previa al modelamiento de la misma.

3. Finalidad

Enseñar y practicar la manera de emplear los Esquemas Preconceptuales [3] y su sintaxis básica, la cual se muestra en la Figura 1, durante la fase de análisis en el desarrollo de una pieza de software orientada por objetos.

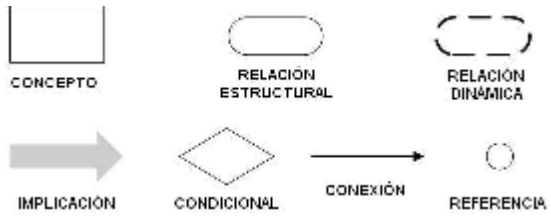


Figura 1. Sintaxis básica de los Esquemas Preconceptuales.

4. Alcance

Este tutorial se limita a la obtención de tres diagramas de UML: clases, comunicación y máquina de estados, tomando como punto de partida los Esquemas Preconceptuales. No se identifican todos los elementos de los tres diagramas anotados, sino únicamente las primitivas conceptuales básicas de cada uno de ellos, las cuales se muestran en la Figura 2. Una descripción más amplia de los tres diagramas se puede apreciar en [2]. Tampoco se proponen otros diagramas UML, puesto que se eligió un diagrama de cada tipo (estructura, interacción y comportamiento), como una manera de ejemplificar el proceso de obtención de cada uno de ellos. Es posible, sin embargo, encontrar reglas para la obtención de otros diagramas, como el de casos de uso o el de actividades.

Ahora, si bien se proponen algunas reglas básicas de traducción entre Esquemas Preconceptuales y los tres diagramas UML mencionados, que se ejemplifican

[1] Este tutorial se realizó en el marco de los siguientes proyectos de investigación: "CONSTRUCCIÓN AUTOMATICA DE ESQUEMAS CONCEPTUALES A PARTIR DE LENGUAJE NATURAL", financiado por la DIME y "DEFINICIÓN DE UN ESQUEMA PRECONCEPTUAL PARA LA OBTENCIÓN AUTOMÁTICA DE ESQUEMAS CONCEPTUALES DE UML" y "CONSTRUCCIÓN DE UN EDITOR GENÉRICO PARA ELABORACIÓN DE MODELOS GRÁFICO SIMBÓLICOS, ETAPA III, Generación de un mecanismo para la Transformación entre Modelos", financiados por DINAIN y administrados por la DIME

en la Tabla 1, también es posible que se discutan otras reglas complementarias.

Figura 2. Primitivas conceptuales básicas de los diagramas de clases, comunicación y máquina de estados, que se identifican a partir de la representación de un discurso en Esquemas Preconceptuales.

5. Resumen

El desarrollo de una pieza de software suele iniciar con un conjunto de entrevistas entre interesados y analistas, con el fin de descubrir las necesidades y expectativas de los interesados y poderlas plasmar en los esquemas conceptuales que representan la solución. Este proceso, sin embargo, está plagado de inconvenientes que surgen por los problemas de comunicación que se originan por las diferencias en las especialidades de los protagonistas del desarrollo de la pieza de software: los interesados, cuyo énfasis fundamental es el dominio del problema, y los analistas, cuyos conocimientos se centran en las herramientas de modelamiento.

Sin importar cuál sea el método de desarrollo que se emplee para la pieza de software, es necesario conciliar estas dos posiciones complementarias, pero cuyos lenguajes son altamente disímiles. Los interesados, por ejemplo, se suelen expresar en lenguaje natural o en documentos de la organización para la que trabajan, en tanto que los analistas se suelen expresar en lenguajes de modelamiento, de los cuales el UML es el principal representante actualmente.

En la Escuela de Sistemas de la Universidad Nacional de Colombia, sede Medellín, se ha propuesto una forma de captura de la información concerniente al dominio del problema: los Esquemas Preconceptuales. Con estos esquemas, es posible realizar dos funciones que se suelen realizar de forma manual en el desarrollo de software:

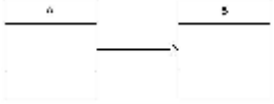
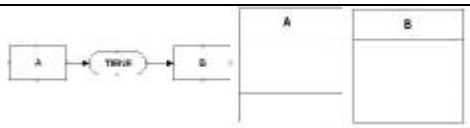
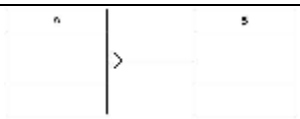
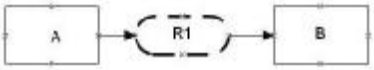
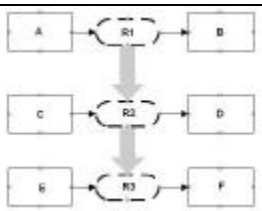
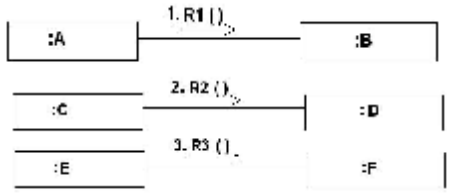
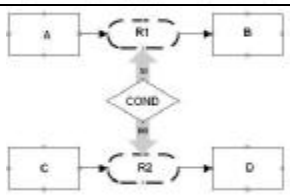
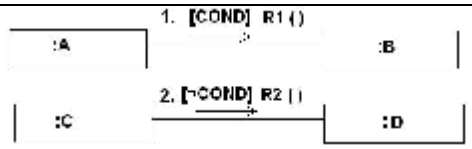
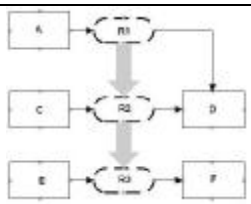
- La representación del dominio del problema en un lenguaje gráfico entendible por los interesados y, consecuentemente, posible de validar por ellos mismos.
- La conversión de esa representación del dominio en tres de los diagramas pertenecientes a UML: clases, comunicación y máquina de estados.

En cuanto a la primera función, es posible que la construcción de esa representación del dominio se realice de manera conjunta entre los analistas y los interesados, con el fin de que se incorporen los conceptos del dominio por parte del interesado y se concilien con la visión técnica del analista, lo cual permite un establecimiento temprano de los requisitos del software y una validación de los mismos por parte de los interesados. En lo relativo a la segunda función, la cual suele ser realizada en forma manual o con asistencia de las herramientas CASE únicamente para efectos de trazado de los diagramas de UML, los esquemas preconceptuales suministran apoyo en la concepción de los diagramas mismos, procurando la automatización de tareas normalmente desarrolladas por los analistas.

En este tutorial se pretende enseñar la sintaxis básica de los esquemas preconceptuales y las bases teóricas de su transformación en los tres diagramas mencionados del estándar UML 2.0. Posteriormente, se construirán algunos de casos de prueba de manera conjunta con los participantes en el tutorial y se mostrará el funcionamiento del prototipo de la herramienta UNC-Diagramador para la elaboración de esquemas preconceptuales y su conversión a los diagramas de UML.

Es importante anotar que el público para este tutorial puede ser diverso, variando desde interesados en el desarrollo de una pieza de software (incluyendo todas las gamas y niveles: profesionales, técnicos, auxiliares, etc.) hasta analistas y desarrolladores. Igualmente, este tutorial se convierte en una herramienta didáctica que puede ser empleada para entender fundamentos de modelamiento de software, por lo cual es importante la asistencia de estudiantes de diversos dominios del conocimiento.

Tabla 1. Expresión Gráfica de las Reglas de obtención de los diagramas de UML a partir de los EPs.

No.	*Precondición*	*Resultado*
1		
2		
3		
4		
5		
6		
7		
8		

Referencias

[1] OMG, UML 2.0 Specification, www.omg.org/UML

[2] M. Fowler, *UML Distilled: A brief guide to the Standard Object Modeling Language*, Addison-Wesley, Boston, 2004.

[3] C. M. Zapata, A. Gelbukh, y F. Arango, "Pre-conceptual Schema: a UML Isomorphism for Automatically Obtaining UML Conceptual Schemas", Research in Computing Science: Advances in Computer Science and Engineering, Vol. 19, 2006, pp. 3–13.

[4] F. Arango y C. M. Zapata, *UN-MÉTODO para la Elicitación de Requisitos de Software*, Carlos Mario Zapata (Ed.), Medellín, 2006.

Como Organizar um Processo Fabril de Produção de Software

José Augusto Fabri

José Augusto Fabri é Doutorando pelo Departamento de Engenharia de Produção da Escola Politécnica da Universidade de São Paulo. Trabalha, ativamente, no desenvolvimento de fábricas de software experimentais e já ministrou várias conferências sobre o TEMA na Europa, Estados Unidos e América Latina. Atua como Professor na Faculdade de Tecnologia de Ourinhos e na Fundação Educacional do Município de Assis.

Marcelo S. de Paula Pessôa

Marcelo S. de Paula Pessôa é Doutor pelo Departamento de Engenharia de Elétrica da Escola Politécnica da Universidade de São Paulo. Trabalha, ativamente, na certificação de processo ISO e CMMI. Membro do comitê internacional da ISO. Possui uma vasta experiência internacional sobre CMMI e certificação de processos de produção de software. Atua como professor do Departamento de Engenharia de Produção da Escola Politécnica da Universidade de São Paulo

fabri@femanet.com.br mpessoa@terra.com.br

Informações Gerais sobre o Tutorial

Apresentar uma estrutura formal que possibilite a organização de Fábricas de Softwares.

Duração: 4 horas

Alcance: Gerentes empresariais e de projetos, desenvolvedores, engenheiro de software, analistas de sistemas e engenheiros de sistemas.

Contextualização do Trabalho

Atualmente, o mercado de desenvolvimento de software latino-americano está começando a tomar consciência sobre a necessidade de se introduzir modelos qualidade em seu processo de produção. Esta informação pode ser comprovada ao analisar os seguintes números:

- A América do Sul possui 13.306 empresas certificadas na norma ISO 9001 (observação: este número engloba todos os setores, inclusive empresas desenvolvedoras de software)[1]. Tal número equivale a 2,36% do total mundial.
- 645 empresas certificadas na norma ISO 14001 (a mesma observação efetuada para a norma ISO 9001 é valida para a norma ISO 14001)[2]. Tal número equivale a 1,8% do total mundial (Uma distribuição das certificações ISO 9001 e 14001 por país latino-americano pode ser verificada na Tabela 1).
- Cerca de 80 empresas com certificação CMMI. O Brasil lidera o quadro com 39 empresas (8º colocado no mundo), seguido pela Argentina com 15 (14ª colocada no mundo), Chile, Colômbia e México completam a lista (fonte: www.sei.cmu.edu/appraisal-program/profile/pdf/CMMI/2006sepCMMI.pdf).
- 3% do mercado global de outsourcing e de serviços de TI (produção de software) dentro do

[1] Acessado a partir de www.mct.gov.br/sepin – Fonte original, segundo a SEPIN, Site da ISO http://www.iso.ch/iso/en/prods-services/otherpubs/pdf/survey13thcycle.pdf

[2] Acessado a partir de www.mct.gov.br/sepin – Fonte original, segundo a SEPIN, Site da ISO http://www.iso.ch/iso/en/prods-services/otherpubs/pdf/survey13thcycle.pdf

contexto mundial, com um montante de 19 bilhões/ano dentro da cadeia de valor[3].

CERTIFICAÇÕES ISO 9001 POR PAÍS DA AMÉRICA DO SUL		
País da América do Sul	Total de empresas certificadas	%Continente
Argentina	2.260	16,98
Bolívia	31	0,23
Brasil	7.900*	59,37
Chile	327	2,46
Colômbia	1.838	13,81
Equador	34	0,26
Guiana	7	0,05
Paraguai	65	0,49
Peru	270	2,03
Suriname	1	0,01
Uruguai	231	1,74
Venezuela	342	2,57
Total	13.306	100
CERTIFICAÇÕES ISO 14001 POR PAÍS DA AMÉRICA DO SUL		
País da América do Sul	Total de empresas certificadas	%Continente
Argentina	175	27,13
Bolívia	3	0,47
Brasil	350	54.26
Chile	17	2,64
Colômbia	41	6,36
Equador	2	0,31
Guiana	3	0,47
Paraguai	1	0,16
Peru	15	2,33
Suriname	0	0,00
Uruguai	29	4,50
Venezuela	9	1,40

Tabela 1 – Certificações ISO 9001 e 14001 dos países latino americano

Analisando o contexto numérico apresentado, é possível constatar um paradoxo em relação aos modelos de qualidade no setor produtivo genérico (todas empresas) e no setor de software. A Argentina e o Brasil, juntamente, com o Chile, México e Colômbia possuem boas classificações se analisado sobre a ótica do CMMI, porém toda América do Sul não possui nem 5% das certificações de qualidade ISO 9001 e 14001. Estas colocações não garantem o acesso a maior fatia do mercado mundial, visto que o nosso continente possui apenas 3% da cadeia de valor comercializada.

Estes números podem "e devem" ser revertidos, principalmente, se analisado dentro do contexto da produção de software. Para que isto ocorra, deve haver um esforço em conjunto das seguintes instituições ou organizações:

- Governo: Melhorando o sistema tributário, incentivando a exportação de produtos relacionados a TI, disponibilizando acesso ao ensino relacionado a engenharia de software as comunidades carentes.
- Empresas: conscientizando sobre a necessidade de certificação em modelos de processos, como o CMMI.
- Universidades: dando ênfase em aspectos da qualidade e produtividade nas disciplinas do núcleo de engenharia de software, despertando, assim, o caráter empreendedorista nos alunos. Temas como: produtividade, qualidade do processo e do produto software (**FÁBRICA DE SOFTWARE)**, devem ser abordados de fortemente nos bancos escolares**.**

As considerações acima apresentadas, motivaram o desenvolvimento deste trabalho cujos objetivos são:

- Responder as seguintes perguntas: O que é o processo fabril de software? Como implantá-lo quando um ambiente é criado?
- Apresentar a situação do mercado de produção de software na América Latina.
- Apresentar como está organizado o processo de produção de 4 fábricas de software: 2 latinas-americanas e 2 japonesas.
- Apresentar como está organizado o processo de produção de software da Microsoft.
- Apresentar a proposta de um mecanismo de criação de um processo fabril para desenvolvimento de software para as empresas.

Para atingir os objetivos propostos, os autores dividiram o tutorial em 9 partes:

- Parte 0: Analise do Mercado Latino-Americano de Produção de Software.
- Parte 1: Apresentação uma revisão bibliográfica sobre fábrica de software.
- Parte 2: Engloba uma revisão bibliográfica sobre processo de software e os mecanismos e modelos que proporcionam a organização ou criação de processos.
- Parte 3: Apresentação das 2 fábricas latinas.

[3] Fonte: *Brazilian Association of Software & Service Export Companies* – BRASSCOM com assessoria da A.T. Kearney (consultoria especializada no desenvolvimento de pesquisa de mercado para a área de Tecnologia da Informação) - http://www.brasscom.com.br

- Parte 4: Apresentação das 2 fábricas Japonesas e do processo de produção da Microsoft.
- Parte 5: Comparação das fábricas latinas com as fábricas japonesas.
- Parte 6: Proposta do mecanismo de criação de um processo fabril para desenvolvimento de software para as empresas.
- Parte 7: Aplicação do mecanismo em um caso real.
- Parte 8: Conclusões e discussões com a audiência.

Referências Bibliográfica

[1] Avrillionis, D.; Conradi, R.; Cunin, P-Y.; Nguyen I. R. (1999); Meta-Process. In: DERNIAME, Jean-Claude; KABA, Badara A.; WASTELL, David. **Software Process: Principles, Methodology, and Technology**. Series: Lecture Notes in Computer Science, Vol. 1500.

[2] Conradi, R.; Jaccheri, M. L.; Process Modeling Language. In: DERNIAME, Jean-Claude; KABA, Badara A.; WASTELL, David. **Software Process: Principles, Methodology, and Technology**. Series: Lecture Notes in Computer Science, Vol. 1500. 1999.

[3] Cusumano, M A. Software Factory: A Historical Interpretation. **IEEE Software**, v. 6. n. 2. p. 23-30. 1989.

[4] Cusumano. M. A. **Japan's Softwares Factories**. New York: Oxford University Press. 1991.

[5] CUSUMANO M. A.; SELBY R. W. How Microsoft Builds Software. **In: ACM Comunication**. n. 40, p. 53-51. doi=http://doi.acm.org/101145/25556.255698. June, 1997.

[6] Feiler, P.; Humphrey, W. 1993 Software Process Development and Enactment: Concepts and Definitions. In: **International Conference on the Software Process**, ICSP Proceedings. Berlin, Germany: IEEE Computer Society Press. 1993.

[7] Reis, R. Q. **APSEE: Um Meta-Modelo para Apoiar a Reutilização de Processo de Software**. Tese de doutorado apresentada ao Instituto de Informática da Universidade Federal do Rio Grande do Sul. 2002.

[8] Schaefer, Wilhelm; Fuggetta, Alfonse; Gobart Claude, Jahnke Jens. Architectural Views and Alternatives. In: DERNIAME, Jean-Claude; KABA, Badara A.; WASTELL, David. **Software Process: Principles, Methodology, and Technology**. Series: Lecture Notes in Computer Science, Vol. 1500. 1999.

El uso de la incertidumbre como herramienta en la ingeniería de software

Nelson Medinilla Martínez
Facultad de Informática
Universidad Politécnica de Madrid
nelson@fi.upm.es

Abstract

Uncertainty can be harmful in some cases and beneficial in other cases. At the beginning, the software universe only appreciates the beneficial side of uncertainty, although it took advantage of it. Software is practically useless without uncertainty. Nowadays, uncertainty is more used as a tool but this is not recognized and it causes several problems. In that sense, this work proves uncertainty can be used as a tool in order: 1) to simplify descriptive and uncertainty complexity; it means, to reduce development and maintenance effort. 2) To analyze and forecast key properties of software engineering components. 3) To appreciate relations between these components and the harmonic place corresponding to each of them. In short, this work proves uncertainty dimension is a basic element in the current theoretical framework of software engineering.

1. Introducción

Es imposible observar una circunferencia completa en un espacio de una dimensión y también es imposible lograr la coincidencia de un guante derecho con otro izquierdo en un espacio de dos dimensiones. De modo semejante, es imposible resolver innumerables problemas en el universo software tradicional porque sólo contempla una dimensión de complejidad: la *complejidad descriptiva* (cantidad de información para describir el sistema). Es una dimensión útil, pero su luz es insuficiente para iluminar el universo software. Esta penumbra provoca visiones complicadas y contradictorias de los recursos software, de las técnicas de diseño, de las metodologías; dificulta ver las relaciones entre todos ellos y el lugar armónico que le corresponde a cada uno en el universo software. Se confunden los objetos con el estructurado, el Proceso Unificado con la Cascada. Se cree que la modularidad, la cohesión, el acoplamiento y la privacidad de las variables son guías del buen diseño, cuando realmente dicen poco o nada; se cree que copiar la realidad dentro del software es otra guía del buen diseño cuando, en muchos casos, produce consecuencias negativas como la reducción drástica de la facilidad de modificación y de evolución. La escasa luz de la complejidad descriptiva muestra a las metodologías y a las técnicas de diseño como recetas aisladas que dicen representar la mejor práctica, inventada por un gurú o un grupo, mientras otro gurú o grupo dice lo contrario porque también defiende su mejor práctica.

Aunque parezca paradójico, la solución a estos problemas, la visión armónica del universo software, aparece cuando se añade la luz de la incertidumbre[1], un elemento considerado fuente de oscuridad, pero que los humanos han usado como solución desde tiempos inmemoriales. Fig. 1.

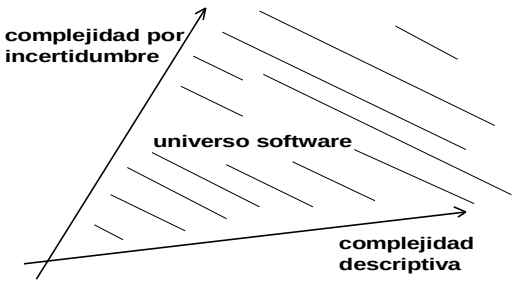


Fig. 1. Ampliación del universo software con la dimensión incertidumbre

La *complejidad de incertidumbre* (cantidad de información necesaria para resolver cualquier incertidumbre asociada con el sistema) agrega otra dimensión al universo software. Fig. 1. Al ampliar las dimensiones del espacio aparecen partes ocultas de los problemas y aparecen también soluciones nuevas, como sucede en los casos geométricos descritos al inicio. La circunferencia sólo muestra uno o dos puntos en una dimensión y aparece completa en dos

[1] *Incertidumbre* es una palabra de amplio contenido semántico: problemático, cuestionable, vago, no definido o determinado, dudoso, no seguro, ambiguo, sujeto a oportunidad o cambio, no estable, variable, no confiable. Todos estos significados se pueden agrupar en dos categorías: *vaguedad* (imprecisión) y *ambigüedad*.

dimensiones.

Sin embargo, durante mucho tiempo, la ingeniería de software ha considerado la incertidumbre perjudicial y erradicable. Por tanto, ha ignorado que la usa constantemente como recurso de simplificación, que es inherente al diseño y además, ha deseado eliminarla aplicando métodos exactos y completos sobre requisitos exactos y completos. Así, dada la erradicabilidad de la incertidumbre y infalibilidad de los métodos, si algo falla, la culpa es de los veleidosos clientes o del uso inadecuado del método o que el tiempo fue insuficiente. Dicho en otras palabras, si la realidad no se ajusta al modelo la culpa es de la realidad. Los trabajos seminales de Dijkstra, Parnas, los tipos abstractos de datos y otros más, guardan graves cicatrices de esa guerra contra la incertidumbre porque proponían soluciones que aprovechan la incertidumbre, aunque no lo supieran.

Actualmente, hay diversos trabajos que aceptan la incertidumbre en la ingeniería de software como problema inevitable, pero que no la utilizan como solución, ni dicen la manera de hacerlo, por ejemplo. Y hay muchos trabajos que utilizan la incertidumbre como solución, pero que no lo dicen o no lo saben y se enredan en explicaciones con terminología inadecuada.

El presente tutorial revisa el universo software a la luz de la dimensión de incertidumbre con el objetivo de demostrar la utilidad de la incertidumbre como herramienta en la ingeniería de software.

2. La incertidumbre como herramienta

2.1 En las abstracciones

Es difícil determinar desde cuando los humanos utilizan la incertidumbre a su favor, pero sin duda la usaron cuando inventaron las abstracciones. Una abstracción es un recurso de simplificación que expresa sólo lo esencial de algo y omite el resto. Esta omisión introduce deliberadamente incertidumbre que se manifiesta en forma de ambigüedad. Una abstracción puede ser precisa respecto a la esencia que expresa, pero es necesariamente ambigua respecto a las alternativas que representa. Sin embargo, no se asocia abstracción con ambigüedad a pesar de ser la ambigüedad el recurso principal de la abstracción.

La evolución del software está marcada por el aumento sistemático de la capacidad para expresar ambigüedad, como recurso para aumentar su capacidad de enfrentar problemas complejos. Primero las *variables simples*, después los *vectores* y *tablas*, más tarde las *listas*. En algún momento, aparecieron las *funciones* y *procedimientos* para simplificar la complejidad, descriptiva y de incertidumbre, mediante las abstracciones que representan sus cabeceras. Cuando el nivel de ambigüedad fue insuficiente se inventaron los *tipos abstractos de datos* para operar con los datos de forma ambigua. Ninguna parte del sistema toca un tipo abstracto de dato, salvo las funciones específicas destinadas a manejarlo. También por los años sesenta, se inventaron los *objetos*, un enfoque radicalmente distinto al estructurado, con una capacidad de ambigüedad superior a todo el software precedente. Por un lado, los *objetos* que expresan *cosas* que se relacionan y por otro, los *mensajes* que expresan comunicaciones variables. Las *clases*, las *clases abstractas*, el *polimorfismo*, etc. aumentan todavía más la capacidad de ambigüedad y, consecuentemente, la capacidad para abordar problemas más complejos en el sentido descriptivo y de incertidumbre.

Los objetos y el estructurado son dos enfoques absolutamente distintos, por la forma de pensar el software (cosas en vez de funciones y datos), y por la capacidad de expresar ambigüedad. Esta última cualidad es la promotora principal de los objetos y también es la cualidad que permite vestir de objetos a diseños estructurados, con la consiguiente pérdida de las ventajas de los objetos y del estructurado. El arraigo del pensamiento estructurado trata de sobrevivir en el contexto de los objetos, y se aprovecha de la ambigüedad potencial de los objetos para esconderse en los ropajes.

2.2 En el diseño

La incertidumbre es inherente a la actividad de diseño, sea o no conocido el problema, por dos causas. Primero, porque está presente en cualquier aspecto novedoso, por nimio que sea. Algo es nuevo sólo si es desconocido antes que aparezca, aunque sea sólo para quién lo percibe. Segundo, porque hay incertidumbre en todas las decisiones de diseño y en cada decisión hay que elegir una alternativa; en cada decisión hay riesgo. La magnitud de la incertidumbre asociada con el diseño influye sobre su proceso de realización.

La incertidumbre también se utiliza en el diseño como herramienta de trabajo, pero se confunde con otras técnicas, sobre todo con “divide y vencerás”. Un diseño software es un producto creativo donde se ha utilizado la división y la ambigüedad como recursos para simplificar la solución. La ambigüedad se manifiesta en las abstracciones y en sus relaciones.

La división no es una herramienta universal y todopoderosa. La división provoca dificultades en universos no aditivos, donde la suma de las partes es desigual del todo, como sucede a menudo en el software. Además, la división es incapaz de simplificar

la complejidad de incertidumbre. Sin embargo, la técnica de introducción de de incertidumbre es capaz de simplificar la complejidad descriptiva y de incertidumbre, aún en universos no aditivos porque no divide. Por ejemplo, la cantidad de información que se necesita para usar una función software se limita a la cabecera de la función y esta cantidad de información se mantiene igual a pesar de la realización de cambios en la implementación, siempre que se conserve la cabecera (abstracción). La incertidumbre asociada con los cambios no altera la cantidad de información gracias a la ambigüedad. Por tanto, se facilitan los cambios en el software.

2.3 En las estrategias de resolución

Los humanos utilizan diferentes estrategias según la magnitud de la incertidumbre que deben enfrentar. Emplean una estrategia lineal o industrial, cuando la incertidumbre es nula o despreciable; una estrategia cíclica o experimental, cuando la incertidumbre es media (se conoce algo) y una estrategia exploratoria o de ensayo y error, cuando la incertidumbre es alta. Mientras mayor sea la incertidumbre de la estrategia mayor será su capacidad para enfrentar incertidumbre.

La *estrategia lineal* (un paso tras otro) lleva de un punto inicial a un punto final, siempre que ambos puntos y el camino sean perfectamente conocidos. Es decir, se tiene que conocer el problema, la solución y la forma de llegar a ella. Si se cumplen todas estas condiciones, la estrategia lineal es la más económica. Para que se cumplan las condiciones hay que erradicar la incertidumbre antes de comenzar la resolución. El paradigma de estrategia lineal en el software es la *Cascada*, que sigue la secuencia *requisitos*, *análisis*, *diseño*, *implementación* y *pruebas*, una traducción de los preceptos cartesianos del Método: *evidencias*, *análisis*, *síntesis* y *evaluación*. La idea que subyace es "primero *qué* y después *cómo*", un deseo, generalmente, iluso. La llamada *estrategia incremental* es una variante de la estrategia lineal que divide el problema en trozos y los aborda uno a uno.

La *estrategia cíclica o experimental* (aproximaciones sucesivas), si converge, se acerca progresivamente a un punto final desconocido mediante el refinamiento periódico de una proposición (hipótesis) inicial. La estrategia cíclica se utiliza cuando se desconoce la solución, pero se tiene alguna idea que permite hacer una hipótesis. El paradigma de estrategia cíclica en el software es la *Espiral*. A veces se dice que cada ciclo de la Espiral es una Cascada, pero esto es una confusión, porque la Espiral reconoce la presencia de incertidumbre durante todo el proceso (conducido por riesgo) y la Cascada, grande o pequeña, exige erradicar la incertidumbre antes de comenzar.

La *estrategia arbórea o exploratoria* (ensayo y error), si el universo es cerrado, es la vía para llegar a un punto desconocido sin tener siquiera una idea. Si el universo es abierto, la estrategia exploratoria no asegura encontrar la solución, pero tampoco lo asegura ninguna otra estrategia en las mismas condiciones de incertidumbre. La estrategia exploratoria se manifiesta cada vez que se desecha una solución y se retrocede al punto anterior. El ciclo de vida llamado *Caos* se acerca a la estrategia exploratoria, pero la obsesión de su autor, por la Cascada lo limita.

2.4 En el orden

Por último, la incertidumbre también ofrece un panorama armónico de la ingeniería de software, si se utiliza como criterio de orden. Fig. 2.

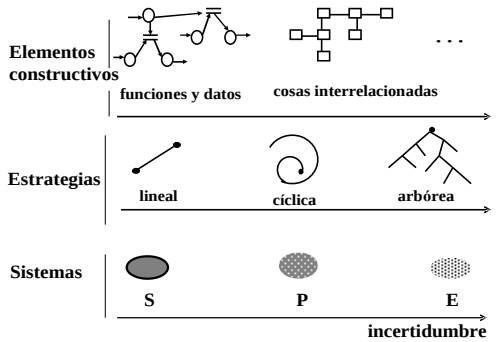


Fig. 2. La incertidumbre como criterio de orden

Puesto que todas las piezas estudiadas de la ingeniería de software expresan o contienen incertidumbre se pueden ordenar en un eje de incertidumbre, por cada tipo de pieza, para compararlas de forma objetiva. Por ejemplo, los objetos tienen una ambigüedad potencial mayor que el estructurado; la estrategia arbórea tiene más incertidumbre que la cíclica. Y de este modo se puede encontrar un lugar para cada una de ellas. Es decir, un espacio armónico donde encajen las distintas piezas de la ingeniería de software y se aprecien las relaciones entre ellas. Las piezas que participan en un proyecto software deben estar en sintonía con la incertidumbre, como afinan los instrumentos de una orquesta con el primer violín o en términos eléctricos, todos deben tener la misma impedancia.

Aplicación de Técnicas de Aprendizaje Cooperativo en la Enseñanza del Desarrollo de Software

Pedro Campos[1], Luis Alberto Flores[2], José Antonio Pow-Sang[2] y Claudia Zapata[2]
[1]*Universidad del Bío-Bío, Chile*
pgcampos@ubiobio.cl
[2]*Pontificia Universidad Católica del Perú, Perú*
{luis.flores,japowsang,zapata.cmp}@pucp.edu.pe

Resumen

El desarrollo de software requiere de algunas destrezas que se desarrollan con la práctica y el trabajo en equipo. Es importante contar con una buena base teórica pero no suficiente, es por esta razón que las técnicas del aprendizaje cooperativo juegan un papel muy importante como herramientas que permiten a través de sus ejercicios aprender nuevos conceptos, afianzar los conceptos ya aprendidos y desarrollar habilidades de análisis frente a una problemática

En este tutorial de dos horas se expondrá diversas experiencias de aplicación de técnicas de aprendizaje cooperativo como rompecabezas, estudio de casos, rally y proyecto en grupo en pre-grado sobre temas de análisis, diseño e implementación orientado a objetos en la Pontificia Universidad Católica del Perú desde el año 2004..

1. Introducción

Cada vez que nos enfrentamos a un grupo de alumnos y pretendemos enseñarles conceptos sobre desarrollo de software encontramos en muchos de ellos gran desinterés y muchas veces no sabemos si en realidad está escuchando y absorbiendo por lo menos la mitad de lo que estamos tratando de explicarles. Normalmente una clase expositiva sobre desarrollo de software no es motivadora para un alumno pues este relaciona el aprendizaje del desarrollo de software con una experiencia puramente práctica sin tomar conciencia de la necesidad de tener conceptos formales al respecto.

De nada nos servirá intentar ser oradores perfectos y divertidos para impartirles a nuestros alumnos conceptos formales si no se acompaña esto con experiencias que permitan integrar sus conocimientos y sus habilidades en la solución de situaciones problemáticas que permitan entrenar al estudiante para su desempeño profesional.

Esto es claro ya que en el desarrollo de software los objetivos no sólo son conocer, reflexionar y explicar sino aplicar los conocimientos aprendidos y las habilidades de análisis y construcción de software para obtener una solución adecuada a una problemática real.

En el desarrollo de software es crítica la etapa de entendimiento del problema y la identificación de las necesidades a satisfacer, por esta razón es necesario que la persona relacionada a dicha actividad cuente con habilidades de comunicación, abstracción y análisis para lo cual las técnicas de aprendizaje cooperativo son de gran utilidad.

Los proyectos de desarrollo de software constan de diferentes personas desempeñando un rol específico y trabajando en equipo dirigidos a un mismo objetivo para lo cual requieren de una interacción efectiva y armoniosa. El aprendizaje cooperativo ayuda de manera significativa en la preparación de los futuros profesionales para poder trabajar en equipos.

El aprendizaje cooperativo involucra el trabajo en grupos desarrollada dentro o parte fuera del salón de clase. El trabajo en equipo permite que ante la deficiencia de algunos alumnos los demás integrantes del grupo lo apoyen a llegar al nivel necesario.

Los métodos que se tratará en este tutorial son:

- Rompecabezas:

El método del rompecabezas como método de enseñanza tiene una estructura de dependencia mutua. Para ejecutar exitosamente una tarea, los alumnos se ven obligados a cooperar, porque cada uno dispone solamente de una parte de la información. Cada uno debe transferir su información a los miembros del grupo y, ya sea individualmente o en conjunto, deben juntar sus piezas de información como un rompecabezas para poder culminar su tarea de manera que tenga sentido. El método del rompecabezas combina de esta manera cooperación con enseñanza mutua

- Estudio de casos:

La técnica de estudio de casos, consiste en proporcionar una serie de casos que representen situaciones problemáticas diversas de la vida real para que se estudien y analicen. De esta manera, se pretende entrenar a los alumnos en la generación de soluciones. Específicamente, un caso es una relación escrita que describe una situación acaecida en la vida de una persona, familia, grupo o empresa. Su aplicación como estrategia o técnica de aprendizaje entrena a los alumnos en la elaboración de soluciones válidas para los posibles problemas de carácter complejo que se presenten en la realidad futura

- Rally:

El trabajo con equipos de alumnos se puede organizar como un rally en el salón. Los grupos de alumnos, en equipos, intentan realizar su mejor prestación. En esta forma es esencial la colaboración dentro de los grupos y la competencia entre ellos. Esta estructura refuerza su compromiso con la meta del grupo y crea con ello un lazo de confianza entre los miembros del equipo (Slavin, 1983)

- Proyecto en grupo:

En el trabajo por proyecto en grupo, los alumnos pueden experimentar y practicar la forma de trabajo en equipo buscando algo nuevo, experimentando y practicando juntos. El proyecto en grupo es la forma más abierta, pero también más compleja del aprendizaje cooperativo (Sharan y otros, 1980) Los alumnos deben ser capaces de trabajar independientemente y en grupos, deben poder formular ideas, hacer planes, repartir tareas, analizar e integrar en conjunto la información recolectada y finalmente presentar sus experiencias a otros.

2. Objetivos y Estructura del Tutorial

Este tutorial está dirigido a docentes de carreras involucradas con el desarrollo de software que deseen incluir sesiones para el aprendizaje cooperativo en sus cursos. Se expondrá cómo se formaron los grupos y cómo se elaboraron los materiales para las experiencias mostradas. También, se presentarán los resultados obtenidos al evaluar el conocimiento de los alumnos antes y después de una clase cooperativa. Y se compartirán las conclusiones y recomendaciones obtenidas de estas experiencias.

A continuación se presenta el programa del tutorial:

- Introducción
  - Fundamentos del Aprendizaje Cooperativo
  - Métodos
- Experiencias
  - Rompecabezas
  - Estudio de casos
  - Rally
  - Proyecto en grupo
- Resultados obtenidos
- Conclusiones y recomendaciones.

10. Referencias

[1] Aronson, E. et al, The Jigsaw classroom, Sage, Beverly Hills, 1978.

[2] Coad, P., North,D.,Mayfield,M.,Object models:strategies, patterns and applications,Prentice-Hall, USA, 1997.

[3] Roeders, P., Un Diseño del Aprendizaje Activo, Walkiria Ediciones con apoyo de la Cooperación Técnica Alemana, primera edición peruana, Lima, 1997.

[4] Dirección de Investigación y Desarrollo Educativo, Vicerrectoría Académica, Instituto Tecnológico y de Estudios Superiores de Monterrey, Estrategias y técnicas didácticas en el rediseño.