
ANALES
36º JAIIO
ISSN 1850-2776
36º Jornadas Argentinas de Informática
ASAI 2007
ASSE 2007
AST 2007
SIS 2007
SID 2007
SSI 2007
SIO 2007
SIE 2007
EST 2007
JSL 2007
JII 2007
JUI 2007
SADIO
Sociedad Argentina de Informática
Mar del Plata
Editores:
Roberto Giordano Lerena
Isabel Pasoni
Pablo Montini

36º Jornadas Argentinas de Informática

36º JAIIO

Simposios

ASAI 2007
ASSE 2007
AST 2007
SIS 2007
SID 2007
SSI 2007
SIO 2007
SIE 2007
EST 2007
JSL 2007
JII 2007
JUI 2007

Auspicios

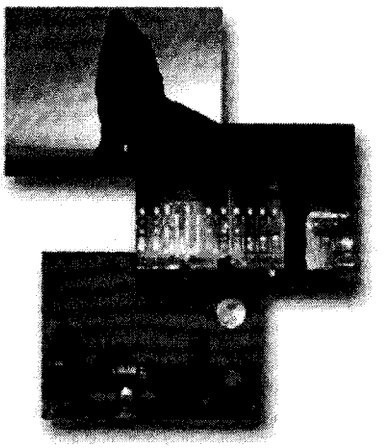
Organizan

Sociedad Argentina de Informática

01 CAECE FASTA

36º Jornadas Argentinas de Informática

36º JAIIO



27 al 31 de Agosto 2007

Mar del Plata, Argentina

Organiza

Sociedad Argentina de Informática

Organizadores Locales

UNIVERSIDAD CAECE Mar del Plata

UNIVERSIDAD FASTA

Simposio Argentino de Ingeniería de Software

Chairs

Alejandro Bianchi
(Liveware)

Alejandra Cechich
(Universidad Nacional del Comahue)

Miembros del comité

- **Adriana Martín** (GIISCo, UNComa, Argentina)
- **Agustina Buccella** (GIISCo, UNComa, Argentina)
- **Alicia Díaz** (LIFIA, UNLP, Argentina)
- **Alejandra Cechich** (GIISCo, UNComa, Argentina)
- **Analía Amandi** (ISISTAN, UNCPBA, Argentina)
- **Andrés Flores** (GIISCo, UNComa, Argentina)
- **Angeles S. Places** (U A Coruña, España)
- **Antonio Rito-Silva** (U Técnica de Lisboa, Portugal)
- **Antonio Vallecillo** (U Málaga, España)
- **Auri Vincenzi** (U. Federal de Goiás, Brasil)
- **Aurora Vizcaíno** (UCLM, España)
- **Carina Alves** (U Federal do Pernambuco, Brasil)
- **Claudia Pons** (LIFIA, UNLP, Argentina)
- **Daniel Riesco** (UNSL, Argentina)
- **Elsa Estévez** (UNS, Argentina)
- **Ernesto Pimentel** (U Málaga, España)
- **Félix García** (UCLM, España)
- **Filippo Lanubile** (U Bari, Italia)
- **Gabriela Aranda** (GIISCo, UNComa, Argentina)
- **Germán Montejano** (UNSL, Argentina)
- **Hernán Astudillo** (UTF Santa María, Chile)
- **Jaelson Castro** (U Federal do Pernambuco, Brasil)
- **Luca Cernuzzi** (U Católica "Nuestra Señora de la Asunción", Paraguay)
- **Luis Olsina** (UNLa Pampa, Argentina)
- **Marcelo Campo** (ISISTAN, UNCPBA, Argentina)
- **Marcelo Frias** (FCEFyN, UBA, Argentina)
- **Marcello Visconti** (UTF Santa María, Chile)
- **Maurice ter Beek** (ISTI, CNR, Italia)
- **Mauricio Marín** (U Magallanes, Chile)
- **Nicolás Kicillof** (FCEFyN, UBA, Argentina)
- **Oscar Pastor** (UP Valencia, España)
- **Pablo Fillotrani** (UNS, Argentina)
- **Regina Motz** (U de la República, Uruguay)
- **Sebastián Uchitel** (FCEFyN, UBA, Argentina)
- **Sergio Ochoa** (U de Chile, Chile)
- **Silvia Gordillo** (LIFIA, UNLP, Argentina)
- **Silvia Amaro** (GIISCo, UNCOMA, Argentina)
- **Victor Braberman** (FCEFyN, UBA, Argentina)
- **Xavier Franch** (UP Catalunya, España)

Comité Operativo

- **Alessandro Fantechi** (Universidad de Florencia, Italia)
- **Antonio Bucchiarone** (Siemens-Nokia, Portugal)
- **Arturo Zambrano** (LIFIA, UNLP, Argentina)
- **Fabio Calefato** (Universidad de Bari, Italia)
- **Germán Vázquez** (UNICEN, Argentina)
- **Guillermo Covella** (UNLPampa, Argentina)
- **Hernán Molina** (UNLPampa, Argentina)
- **Magali González** (U Católica "Nuestra Señora de la Asunción", Paraguay)
- **Teresa Mallardo** (Universidad de Bari, Italia)

CIDISI - Centro de Investigación y Desarrollo en Ingeniería en Sistemas de Información,
Universidad Tecnológica Nacional – Facultad Regional Santa Fe - Argentina

- Modelos de Madurez en la Industria del Software: Evaluación de un Modelo para Pequeñas, Medianas Empresas, Alicia Mon, Marcelo Estayno, Andrea Arancio
  Grupo de Ingeniería de Software (G.I.S.)
  Universidad Nacional de La Matanza, Buenos Aires. Argentina
  Nancy Velásquez, Quito, Ecuador

- Desarrollo Orientado a Servicios de Sistemas de Información: un enfoque basado en el modelo de negocio
  Valeria de Castro, Esperanza Marcos
  Grupo de Investigación Kybele Escuela de Ciencias Experimentales y Tecnologías
  Universidad Rey Juan Carlos, España

- Testing Service Composition
  Antonio Bucchiarone, Hernán Melgratti1, Francesco Severoni
  IMT Lucca, Italy
  Dipartimento di Informatica, Universitá di L'Aquila, Italy

- Ingeniería inversa: una experiencia real con beneficios para la madurez del equipo de desarrollo
  Marcelo H. Luna
  Liveware IS, S.A, Buenos Aires, Argentina

- Enterprise Architectures – Enabling Interoperability Between Organizations
  Alejandro Sánchez, Rilwan Basanya, Tomasz Janowski
  Adegboyega Ojo Center for Electronic Governance at United Nations
  University International Institute for Software Technology
  Macau Software Engineering Group, Universidad Nacional de San Luis
  Facultad de Ciencias Físico Matemáticas y Naturales, San Luis, Argentina Department of Computer Sciences at University of Lagos, Faculty of Science, Lagos, Nigeria

- Middleware Web Services para la Interoperabilidad entre Plataformas de Gestión del Aprendizaje
  Julio R. Ribón, Tomás P. de Miguel, Víctor H. Medina
  Universidad de Cartagena, Colombia
  Universidad Politécnica de Madrid, España
  Universidad Distrital Francisco José de Caldas, Colombia

- Modelador y Gestor de Versiones de Procesos Basado en XPDL Jackeline Pedreschi, Sandro Rondón, Meylin Camarena, Carla Basurto, Abraham E. Dávila
  Pontificia Universidad Católica del Perú, Departamento de Ingeniería, San Miguel, Perú

¿Cómo utilizar experimentos controlados en la selección de transformaciones QVT para obtener diagramas de clases UML más fáciles de entender?

José Ángel Carsí[1], Emilio Insfrán[1],
Silvia Abrahão[1], Marcela Genero[2], Mario Piattini[2], Isidro Ramos[1]

[1]Grupo de investigación ISSI,
Departamento de Sistemas Informáticos y Computación,
Universidad Politécnica de Valencia
Camino de Vera s/n, 46022, Valencia
{pcarsi, einsfran, sabrahao, iramos}@dsic.upv.es
[2]Grupo de investigación ALARCOS,
Departamento de Tecnologías y Sistemas de Información, Universidad de Castilla La-Mancha
Paseo de la Universidad Nº 4, 13071, Ciudad Real
{Marcela.Genero, Mario.Piattini}@uclm.es

Resumen. Este trabajo forma parte de un proyecto que estamos desarrollando, cuyo objetivo es la automatización de las transformaciones entre modelos en el contexto de procesos MDA, guiada por atributos de calidad. El presente trabajo presenta un experimento para seleccionar transformaciones guiadas por atributos de calidad cuando existen alternativas. Se centra en un conjunto de reglas de transformación para producir relaciones estructurales entre clases (asociación, agregación y clase asociación) a partir de un modelo de requisitos y el atributo de calidad facilidad de entendimiento. El objetivo del experimento es obtener evidencia empírica sobre qué transformación alternativa produce el diagrama de clases UML que es más fácil de entender. Los resultados demuestran que hay una tendencia en el uso de las relaciones de asociación para guiar estas transformaciones. Sin embargo, más experimentación es necesaria para obtener resultados concluyentes.

Palabras clave: MDA, transformaciones, UML, QVT, diagrama de clase, entendibilidad, experimento

1 Introducción

Model-driven architecture (MDA) es una aproximación emergente que promueve el uso de modelos durante el proceso de desarrollo y permite que éstos puedan ser transformados hasta la obtención del código fuente del producto software final [12]. Por lo tanto, en este enfoque, los modelos ya no son simples medios para describir software, sino la pieza fundamental de su desarrollo. Como consecuencia, la calidad de los modelos tiene una gran importancia, ya que determinará la calidad de los productos software finalmente desarrollados.

En general, hay varias maneras de transformar un modelo origen a un modelo destino. Los modelos destino alternativos pueden tener la misma funcionalidad, pero pueden ser muy distintos en cuanto a sus atributos de calidad. Uno de los desafíos de un proceso de transformación automático es identificar las reglas de transformación que producirán el modelo destino que satisface en mayor grado los atributos de calidad deseados.

En este artículo, se presenta un experimento controlado para investigar la selección de transformaciones alternativas para obtener diagramas de clase UML a partir de un modelo de requisitos [5] [6]. Se centra en un conjunto de reglas de transformación para obtener relaciones estructurales entre clases (asociación, agregación y clase asociación) y el atributo de calidad 'facilidad de entendimiento'. El objetivo del experimento es obtener evidencia empírica sobre qué transformación alternativa produce el diagrama de clase UML que es más fácil de entender.

Las transformaciones para obtener relaciones estructurales entre clases han sido seleccionadas debido a que éstas constituyen la fuente principal de variación e impacto en el diseño cuando se define un diagrama de clases UML. La facilidad de entendimiento ha sido seleccionada porque es el principal atributo de calidad que influencia la mantenibilidad. Un diagrama de clases debe ser bien entendido antes de que cualquier cambio pueda ser identificado o realizado por el analista.

Además, la evaluación empírica de las reglas de transformación es particularmente importante cuando éstas se aplican de forma automática. El conjunto seleccionado de transformaciones se define usando el estándar Query-View-Transformations (QVT) [14] y se ejecuta en la plataforma de gestión de modelos MOMENT [2].

Este artículo se organiza como sigue. La sección 2 discute los trabajos existentes sobre calidad en la transformación de modelos. La sección 3 presenta la aproximación para transformar un modelo de requisitos en diagramas de clases UML. La sección 4 describe el diseño del experimento para validar la selección de las transformaciones alternativas según el atributo facilidad de entendimiento. La sección 5 presenta el análisis de datos y las lecciones aprendidas. Finalmente, la sección 6 presenta las conclusiones y trabajo futuro.

2 Trabajos relacionados

En los últimos años se han propuesto, en el ámbito de la Ingeniería del Software, algunos trabajos que tratan la calidad de las transformaciones desde la perspectiva de algún atributo de calidad. La tabla 1 presenta un resumen de estas propuestas organizadas cronológicamente. Una comparación más detallada se puede encontrar en [1]. La tabla 1 revela que algunas propuestas se centran en definir transformaciones horizontales para el refactoring de modelos [11] [10] [7]. Otras propuestas proporcionan transformaciones verticales para el refinamiento de modelos [15], síntesis [8] [9], o ingeniería inversa [18]. De estos estudios, sólo el de Kurtev [9] presenta una propuesta más sistemática para la selección de transformaciones alternativas en base al atributo de calidad adaptabilidad.

Estas aproximaciones plantean algunos criterios de calidad que se pueden utilizar para dirigir las transformaciones, pero sin embargo, muy pocas ilustran la selección

de transformaciones por medio de ejemplos prácticos [9] [10]. Además, a excepción de Markovic y Baar [10] y Kurtev [9], las transformaciones no están bien definidas. Por lo tanto, son necesarias aproximaciones más sistemáticas para asegurar la calidad de los artefactos software obtenidos como parte de un proceso basado en transformación de modelos.

Tabla 1. Clasificación de propuestas para la calidad de las transformaciones.

Propuesta	Objetivo	Tipo de transformac.	Artefacto de entrada	Atributos de Calidad	Automatizado
Zou y Kontogiannis, 2003 [18]	Ingeniería Inversa (migración)	Vertical (CM-a-PIM)	Programa fuente	Acoplamiento y cohesión	No
Rottger y Zschaler, 2004 [15]	Refinamiento	Horizontal	Modelos de contexto	Tiempo de respuesta	Parcial-mente
Merilinna, 2005 [11]	Refactoring	Horizontal (PIM-a-PIM)	Modelos arquitecturales	Rendimiento, disponibilidad, fiabilidad, mantenimiento y reusabilidad	Si
Kurtev, 2005 [9]	Síntesis	Vertical (PIM-a-PIM)	Diagramas de clase UML	Adaptabilidad	Si
Markovic y Baar, 2005 [10]	Refactoring	Horizontal (PIM-a-PIM)	Diagramas de clase UML	Correctitud semántica	No
Sottet et al., 2006 [16]	–	–	Modelos de Interfaz	Compatibilidad, protección de error, consistencia	No
Ivkovic y Kontogiannis, 2006 [7]	Refactoring	Horizontal (PIM-a-PIM)	Modelos arquitecturales expresados en UML	Mantenimiento, rendimiento y seguridad	No
Kerhervé et al., 2006 [8]	Síntesis, refinamiento	Horizontal y vertical	Modelos de información	Portabilidad de servicios, tiempo de respuesta, retraso de la red	No

(–) indica que la propuesta no proporciona esta información

Otra debilidad de estas propuestas es que la selección de transformaciones alternativas no ha sido validada empíricamente. La aplicabilidad práctica de las transformaciones es comúnmente descrita en base a la intuición de los investigadores. Según lo indicado por Czarnecki y Helsen [3], hay una carencia de experimentos controlados para validar las observaciones realizadas por los investigadores en el campo de la Ingeniería de Modelos. Intentando paliar las carencias mencionadas, en este trabajo, se presenta una propuesta concreta para validar empíricamente la selección de transformaciones alternativas dirigida por atributos de calidad.

3 Transformación de requisitos en diagramas de clases UML

El modelo de requisitos [5] [6] define la estructura y el proceso seguido para capturar los requisitos del software. Se compone de un Árbol de Refinamiento de las Funciones (FRT) para especificar la descomposición jerárquica de las funciones del sistema; un modelo de Caso de Uso (UC) para especificar la comunicación y la funcionalidad; y de Diagramas de Secuencia (SD) para especificar las interacciones entre las entidades para realizar cada caso de uso. La herramienta CASE RETO (http://reto.dsic.upv.es) da soporte a este método. Siguiendo una estrategia MDA de transformación de modelos, una vez especificado el modelo de requisitos, se puede obtener un diagrama de clases UML aplicando un conjunto de reglas de transformación [5].

Una regla de transformación está compuesta de un patrón de condición y una acción. Algunas reglas de transformación son sencillas de aplicar una vez que se ha satisfecho el patrón de condición de la regla (relaciones uno a uno). Por ejemplo, la generación de clases del diagrama de clases UML se basa en el análisis de actores y de las entidades que participan en todos los diagramas de secuencia. Cuando se aplica esta regla de transformación (TR2), una clase es generada en el diagrama de clases para cada entidad de tipo clase distinta que participa en cualquier diagrama de secuencia. Sin embargo, otras reglas no son fáciles de identificar y aplicar debido a tres razones principales: la complejidad del patrón de condición de la regla de transformación, la satisfacción de varias reglas debido a condiciones de transformación no disjuntas, y las posibles múltiples representaciones válidas para un elemento del modelo de requisitos en el diagrama de clases.

El metamodelado es un concepto fundamental del paradigma MDA y es usado en la Ingeniería del Software para describir las abstracciones que definen los modelos y las relaciones entre ellas. Básicamente, un metamodelo se puede representar como un diagrama de clases cuyas clases y asociaciones representan los conceptos y las relaciones entre conceptos. En este contexto, MOF (Meta-Object Facility) [13] proporciona un marco para definir un metamodelo y realizar consultas y manipulaciones sobre el mismo.

A continuación se detallan las partes más relevantes del metamodelo de requisitos. Las funciones del sistema son representadas por la clase UseCase. Cada caso de uso se especifica detalladamente por medio de uno o varios diagramas de secuencia. Los diagramas de secuencia (SequenceDiagram) se componen principalmente de entidades (entity) y mensajes (message). Para describir un diagrama de secuencia, se distinguen tres tipos de entidades: actor, interfaz y clase. El actor representa a usuarios del caso del uso; interfaz representa la frontera entre los actores y las clases del sistema; y clase representan las diversas clases del sistema que participan en la realización del caso de uso.

Finalmente, para caracterizar el tipo de interacción entre objetos, se utilizan cuatro tipos de mensajes: signal, service, query y connect. Los mensajes signal representan la interacción entre los actores y el interfaz del sistema. Los mensajes service representan interacciones entre objetos que tienen el propósito de modificar el estado del sistema (crear, borrar o modificar). Los mensajes query representan interacciones entre objetos necesarias para consultar el estado de un objeto o un conjunto de objetos.

Las reglas de transformación a aplicar para una especificación de requisitos que generan diferentes diagramas de clases. Aunque no todas las posibilidades se pueden explicar con detalle, debido a la limitación de espacio, es posible ver que dado una especificación de requisitos varios diagramas de clase válidos pueden ser obtenidos aplicando distintas reglas de transformación.

El ejemplo usado para ilustrar estas transformaciones alternativas es el de un sistema de alquiler de coches. Un diagrama de secuencia se utiliza para especificar las interacciones entre objetos que son necesarias para realizar el caso de uso Crea Seguro. Este caso de uso es iniciado por un actor Administrador y representa la creación de una póliza de seguro de coche que se contrata con una compañía de seguros para poder alquilar los coches. La Figura 1 muestra el diagrama de secuencia para este caso de uso. Se introducen los datos y se crea un nuevo objeto Seguro que tiene que estar relacionado con su Compañía de Seguros y con el Coche para el que se ha contratado el seguro.

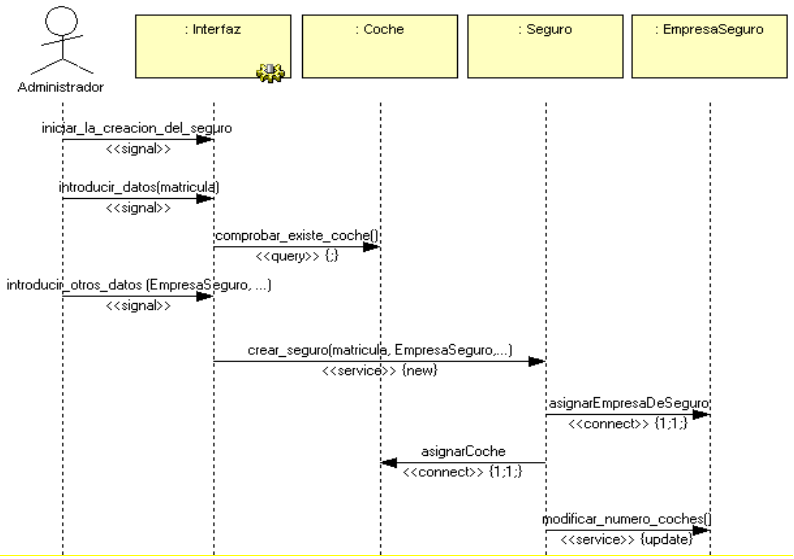


Fig. 1. Diagrama de Secuencia mostrando las interacciones entre objetos (estereotipadas) para el caso de uso Crear Seguro.

Siguiendo una aproximación MDA, se utiliza esta información para transformar la especificación de requisitos en un diagrama de clases. Este proceso de transformación debe decidir qué transformaciones son más convenientes de aplicar para producir el resultado esperado por el analista o un resultado que maximice un atributo de calidad (por ejemplo la entendibilidad).

Analizando el caso de uso Crear Seguro, especificado como interacciones entre clases en la figura 1, se observa que se puede aplicar la regla TR15 de transformación (“Por cada mensaje etiquetado con el estereotipo «connect», generar una relación de asociación entre las clases participantes”) y, consecuentemente, establecer relaciones de asociación entre Seguro y Coche y entre Seguro y Compañía de Seguros como se muestra en la Figura 2a.

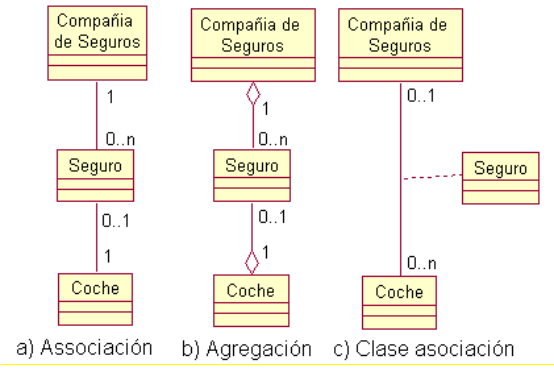


Fig. 2. Parte de diagramas de clases obtenidos por transformación.

Alternativamente, hay otras interpretaciones que pueden realizarse sobre las interacciones entre objetos para la construcción del diagrama de clases. Por ejemplo, una segunda alternativa, podría representar el Seguro como componente en relaciones de agregación con las clases Compañía de Seguros y Coche (Figura 2b, agregación). Una tercera alternativa podría representar el Seguro como una clase asociación entre la Compañía de Seguros y el Coche (Figura 2d, clase asociación).

En definitiva, se han detectado tres tipos de relaciones estructurales para representar estas interacciones entre objetos (por supuesto puede haber otras representaciones válidas) y por tanto, distintas reglas de transformación deben ser aplicadas para producir estas relaciones. En la Tabla 2, se presentan estas relaciones estructurales y las reglas de transformación alternativas que pueden ser aplicadas.

Tabla 2. Relaciones estructurales y sus correspondientes reglas de transformación.

Alternativas	**Reglas de Transformación**
A1 (asociación)	TR 14. Proceso de mensajes «service/new» TR 15. Proceso de mensajes «connect» TR 16. Proceso de mensajes «service/new» o «connect» con clases usando nombre de roles
A2 (agregación)	TR 28. Proceso de mensajes «service/new»
A3 (clase asociación)	TR 39. Proceso de mensajes «service/new» con mensajes «connect» TR 40. Proceso de mensajes «connect» con mensajes «service/new»

4 Descripción del experimento

Esta sección presenta el proceso experimental [17] seguido en la realización del experimento controlado. El principal objetivo del experimento es investigar cual de las alternativas de las reglas de transformación para relaciones estructurales entre

clases introducido en la sección 3.3 (A1, A2, A3) permite obtener el diagrama de clases más fácil de entender. Los participantes del experimento fueron 39 estudiantes de cuarto curso de la Ingeniería Informática de la Universidad Politécnica de Valencia, que estaban cursando un segundo curso de Ingeniería de Software. Se consideraron una "muestra por conveniencia" (es decir todos los estudiantes disponibles en la clase). Los sujetos tenían seis meses de experiencia en modelar con UML y cierta experiencia en el paradigma OO. Los sujetos fueron incentivados a participar ofreciéndoles un punto extra en la nota final por realizar las tareas requeridas correctamente. Las variables independientes son las reglas de la transformación de las relaciones estructurales entre las clases (A1, A2, A3). La variable dependiente es la Facilidad de Entendimiento (Understandability). El material y las tareas experimentales consistieron en:

- 9 diagramas de secuencia sobre tres casos de estudio, con 4 diagramas de la clase UML, cada uno. Cada diagrama de clase fue obtenido aplicando las reglas de transformación alternativas. El material experimental está disponible en: www.dsic.upv.es/~einsfran/experiment.
- Cada diagrama de secuencia tenía un cuestionario adjunto que consistía en 6 preguntas de tipo "Si/No" para comprobar si los sujetos entendían los diagramas de secuencia. La eficacia de los sujetos para contestar las preguntas (número de respuestas correctas/número de respuestas) se utilizó para excluir las observaciones que no satisfacían un nivel mínimo de la calidad. Las observaciones con un valor menor que 0,5 fueron excluidas. Consideramos que si los sujetos no entendían el diagrama de secuencia, no podían realizar el resto de tareas correctamente.
- Cada uno de los tres diagramas de clase UML tenía un cuestionario adjunto (con 6 preguntas) para determinar qué alternativa de diagrama de la clase UML mejor entendido por los sujetos. Además, los sujetos tenían que anotar el tiempo de comienzo y fin para contestar los cuestionarios. De esta tarea se obtuvieron tres medidas para el facilidad de entendimiento:
  - Tiempo de Entendimiento, que refleja el tiempo, en segundos, que los sujetos tardaron en contestar a cada cuestionario (calculado por la diferencia entre la hora de finalización y la hora de comienzo). Cada sujeto contestó tres cuestionarios uno para cada alternativa (A1, A2 y A3), obteniendo así tres medidas del Tiempo Entendimiento (A1Tiempo, A2Tiempo y A3Tiempo).
  - Eficacia, que refleja la corrección de las respuestas (calculadas dividiendo el número de las respuestas correctas por el número de respuestas). Así, se obtuvieron tres medidas de la Eficacia (A2Eficacia, A2Eficacia y A3Eficacia).
  - Eficiencia, que refleja la corrección de las respuestas por unidad de tiempo (calculada dividiendo el número de las respuestas correctas por el Tiempo de Entendimiento). De esta manera, se obtuvieron tres medidas para la Eficiencia (A2Eficiencia, A2Eficiencia y A3Eficiencia).
- La última tarea consistía en preguntar a los sujetos cuál de los tres diagramas de clase UML reflejaba mejor el problema modelado en el diagrama de secuencia. De esta manera, obtuvimos una medida subjetiva (Alternativa Seleccionada) basada en la opinión de los sujetos.

  Se plantearon las siguientes hipótesis:

- $H1_0$: El uso de diferentes transformaciones alternativas (A1, A2, A3) no afecta al Tiempo de Entendimiento (A1Tiempo, A2Tiempo y A3Tiempo). $H1_1 = \neg H1_0$
- Análogamente se definen hipótesis para las variables relacionadas con la Eficacia y la Eficiencia.

El experimento comenzó con una sesión introductoria en la cual se refrescaron los conceptos principales de los modelo de requisitos (es decir, la notación de los diagramas de secuencia). A cada sujeto se le entregó todo el material, con los nueve diagramas de secuencia (diseño balanceado intrasujetos). Los diagramas se entregaron en diferente orden para evitar el efecto de aprendizaje. Se explicó a los sujetos cómo realizar las tareas experimentales, y tuvieron dos horas para realizar todas las tareas.

Después de realizar el experimento, se recogieron los datos del mismo. En primer lugar, se hizo una "limpieza de datos", excluyendo aquellas observaciones que no estaban completas porque no habían anotado el tiempo o porque no habían seleccionado la mejor alternativa. Todas las preguntas fueron contestadas en cada cuestionario, asegurando así la completitud de las tareas realizadas. Por otra parte, se excluyeron las observaciones que tuvieron, para cada diagrama de secuencia, un valor de eficacia menor que 50%. Finalmente, se consideraron 323 observaciones para contrastar las hipótesis.

5 Análisis de datos

La transformación más seleccionada por los sujetos fue A1, como muestra la figura 3. Esto significa que los sujetos creyeron que el uso de asociaciones permitió obtener el mejor diagrama de clases (el más fácil entender). Las clases de asociación, representadas por la alternativa A3, fue la menos elegida, hecho que revela que puede ser la alternativa de alternativa menos apropiada.

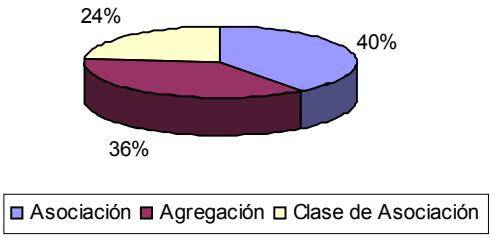


Fig. 3. Porcentajes por Alternativa Seleccionada.

Los estadísticos descriptivos para el Tiempo de Entendimiento, la Eficacia y la Eficiencia muestran que en promedio, los sujetos tardaron menos en realizar las tareas relacionadas con la alternativa A2; aunque realmente la diferencia con las otras no es muy significativa En promedio, los sujetos fueron más eficaces y eficientes realizando las tareas relacionadas con la alternativa A1, pero las diferencias en la eficacia con las otras alternativas no son demasiado muy significativas.

Para contrastar las hipótesis presentadas en la sección 4 se realizó un ANOVA para medidas repetidas. Los resultados obtenidos permitieron rechazar la hipótesis $H1_0$, $H2_0$, y $H3_0$ (con un nivel de significación = 0,05), lo que significa que el uso de diferentes transformaciones alternativas realmente afecta al Tiempo de Entendimiento, a la Eficacia y a la Eficiencia.

Resumiendo, los resultados principales obtenidos a partir de este experimento son:

- Los estadísticos descriptivos muestran una leve tendencia a favor de A1, la transformación basada en asociaciones. Sorprendentemente, la diferencia entre los valores mínimos y máximos del Tiempo es muy significativa. Hecho que puede ser debido a que los sujetos eran modeladores novatos.
- La transformación más seleccionada por los sujetos, es la que se basa en asociaciones.

6 Conclusiones y trabajos futuros

En este artículo se ha presentado cómo es posible utilizar experimentos controlados para seleccionar las transformaciones más adecuadas, cuando en un proceso de transformación existen varias alternativas posibles.

El experimento se diseñó para averiguar qué alternativa era la que mejoraba la facilidad de entendimiento de los diagramas de clases obtenidos a través de transformaciones, partiendo del modelo de requisitos. Se tuvieron en cuenta sólo aquellas transformaciones relacionadas con las relaciones estructurales entre clases.

El resultado más relevante es que la alternativa basada en asociaciones (A1) es la más efectiva y eficiente considerando tanto las medidas objetivas como las subjetivas de la facilidad de entendimiento. Una posible razón para ello podría ser la menor carga semántica que tiene la asociación frente al resto de relaciones.

Una limitación de este estudio es que las tres transformaciones alternativas no pueden usarse simultáneamente en todas las situaciones de modelado. Por ejemplo, para establecer una clase asociación (A4) es necesario tener al menos un mensaje «service/new» y dos mensajes «connect» en el modelo origen. El objetivo del experimento realizado en este trabajo fue obtener evidencia empírica cuando las tres alternativas para un caso específico pueden ser utilizadas indistintamente para obtener una relación entre clases. Somos conscientes de que hay más de tres alternativas para representar relaciones estructurales entre clases. Es necesario realizar más experimentos con estas y otras alternativas.

Otra limitación es el uso de estudiantes como sujetos experimentales. En general los estudiantes no tenían experiencia práctica en modelado conceptual con UML. Este hecho puede afectar la validez externa del experimento. Sin embargo, los sujetos fueron estudiantes de cuarto curso de Ingeniería Informática, por lo podemos considerarlos analistas novatos.

Estos resultados aunque preeliminares proveen evidencia empírica que servirá para definir un modelo de calidad para dirigir la transformación de modelos. Se va a replicar el experimento con otros estudiantes de la Universidad de Castilla-La Mancha y modeladores con más experiencia. Una mayor experiencia en modelado podría influir en el rendimiento de los sujetos. Como trabajo futuro se presente evaluar el

resto de transformaciones del catálogo de reglas [5] considerando otros atributos de calidad (p.e., usabilidad, modificabilidad).

Agradecimientos. Este trabajo es parte de los proyectos META TIN2006-15175-C05-05 y la red CALIPSO (TIN20005-24055-E) ambos financiados por la Ministerio de Educación y Ciencia (España).

Referencias

1. Abrahão S., Genero, M., Insfran E., Carsí J.A., Ramos I., Piattini M.: Quality-Driven Model Transformations: From Requirements to UML Class Diagrams Model-Driven Software Development: Integrating Quality Assurance, Idea Group (2007).
2. Boronat A., Carsí J.A., Ramos I.: Algebraic Specification of a Model Transformation Engine. Proc. of the Fundamental Approaches to Software Engineering (FASE'06). ETAPS'06. Vienna, Austria, (2006) 262–277.
3. Czarnecki K., Helsen S.: Feature-based survey of model transformation approaches. IBM Systems Journal 45(3), (2006) 621–645.
4. Gómez A., Boronat A., Hoyos L., Ramos I., Carsí J.Á.: Definición de operaciones complejas con un lenguaje específico de dominio en Gestión de Modelos, XI Jornadas de Ingeniería del Software y Bases de Datos. JISBD'06. Octubre, 2006. Sitges, Barcelona.
5. Insfran E.: A Requirements Engineering Approach for Object-Oriented Conceptual Modeling, PhD Thesis, DSIC, Valencia University of Technology (2003).
6. Insfran E., Pastor O. and Wieringa R.: Requirements Engineering-Based Conceptual Modelling. Requirements Engineering, 7(2), (2002) 61–72, Springer-Verlag.
7. Ivkovic I., Kontogiannis K.: A Framework for Software Architecture Refactoring using Model Transformations and Semantic Annotations, Proc. of the Conference on Software Maintenance and Reengineering, CSMR 2006.
8. Kerhervé B., Nguyen K.K., Gerbé O., Jaumard B.: A Framework for Quality-Driven Delivery in Distributed Multimedia Systems, Proc. of the Advanced Int. Conference on Telecommunications and Int. Conference on Internet and Web Applications and Services (AICT/ICIW 2006), (2006) 195–205.
9. Kurtev I.: Adaptability of Model Transformations. PhD Thesis, Univ. of Twente, (2005).
10. Markovic S., Baar T.: Refactoring OCL annotated UML class diagrams. 8th Int. Conference on Model Driven Engineering Languages and Systems (2005) 280–294.
11. Merilinna J.: A Tool for Quality-Driven Architecture Model Transformation. Espoo, VTT Electronics, VTT Publications (2005).
12. OMG: MDA Guide. Version 1.0.1, (2003).
13. OMG: Meta Object Facility (MOF) 2.0 Core Specification, ptc/04-10-15 (2004).
14. OMG: QVT MOF 2.0 QVT Final Adopted Specification, OMG 2005.
15. Rottger S., Zschaler, S.: Model-Driven Development for Non-functional Properties: Refinement Through Model Transformation, The Unified Modelling Language Conference, LNCS Volume 3273, (2004) 275–289.
16. Sottet J. S., Calvary G., Favre J.M.: Mapping Model: A First Step to Ensure Usability for sustaining User Interface Plasticity, Proc. of the Workshop on Model Driven Development of Advanced User Interfaces (2006).
17. Wohlin, C., P. Runeson, M. Hast, M. Ohlsson, B. Regnell, A. Wesslen, Experimentation in Software Engineering: an Introduction, Kluwer, 2000.
18. Zou Y., Kontogiannis K.: Quality Driven Transformation Framework for OO Migration. Proc. 2nd ASERC Workshop on Software Architecture, Banff, Canada (2003) 18–19.