Universidad de Murcia
Departamento de Informática y Sistemas

Una propuesta para el tratamiento de requisitos y modelos de calidad en *Model Driven Engineering*

Tesis Doctoral

Doctorando: Fernando Molina Molina

Director: J. Ambrosio Toval Álvarez

2010

University of Murcia
Department of Informatics and Systems

A proposal for the alignment of requirements and quality models in Model Driven Engineering

PhD. Thesis

PhD candidate: Fernando Molina Molina

Supervisor: J. Ambrosio Toval Álvarez

2010

D. José Ambrosio Toval Álvarez, Catedrático de Universidad del Área de Lenguajes y Sistemas en el Departamento de Informática y Sistemas, AUTORIZA:

La presentación de la Tesis Doctoral titulada "Una propuesta para el tratamiento de requisitos y modelos de calidad en Model Driven Engineering", realizada por D. Fernando Molina Molina, bajo mi inmediata dirección y supervisión, y que presenta para la obtención del grado de Doctor por la Universidad de Murcia.

En Murcia, a 17 de Diciembre de 2009

D. Jesús J. García Molina, Catedrático de Universidad del Área de Lenguajes y Sistemas Informáticos y Director del Departamento de Informática y Sistemas, INFORMA:

Que la Tesis Doctoral titulada "Una propuesta para el tratamiento de requisitos y modelos de calidad en Model Driven Engineering", ha sido realizada por D. Fernando Molina Molina, bajo la inmediata dirección y supervisión de D. J. Ambrosio Toval Álvarez, y que el Departamento ha dado su conformidad para que sea presentada ante la Comisión de Doctorado.

Murcia, a 17 de Diciembre de 2009

Detalles formales de presentación

Esta tesis doctoral opta a la Mención de Doctorado Europeo y se presenta en la modalidad de *compendio de publicaciones* gracias a la autorización, que se adjunta, emitida en este sentido por la Comisión General de Doctorado de la Universidad de Murcia. La normativa de la modalidad de *compendio de publicaciones* únicamente obliga a incluir como documentación de la tesis un breve resumen de la misma y al menos tres artículos publicados, o aceptados para su publicación, en revistas indexadas de reconocido prestigio, como son las incluidas con factor de impacto en el listado ISI/JCR (*Journal Citations Report*). A continuación, se muestran las referencias a estos tres artículos y la de un cuarto en el que también ha estado implicado el doctorando durante la elaboración de esta tesis, pudiendo ser consultados en el Apéndice *'Publicaciones'* de este documento:

- Molina, F. and Toval, A (2009). Integrating usability requirements that can be evaluated in design time into Model Driven Engineering of Web Information Systems. Journal of Advances in Engineering Software, Vol. 40, Issue 12, pp. 1306-1317. **Factor de impacto en 2008: 1.188**
- Molina, F., Pardillo, J., Cachero, C. and Toval, A (2010). An MDE Modelling Framework for Measurable Goal-oriented Requirements. International Journal of Intelligent Systems. Aceptado para su publicación en el Special Issue 'Goal-Driven Requirements Engineering'. **Factor de impacto en 2008: 0.860**
- Sánchez, O, Molina, F., García-Molina, J. and Toval, A (2010). ModelSec: a generative architecture for model-driven security. Journal of Universal Computer Science. Aceptado para su publicación en el Special Issue 'Security in Information Systems: New Advances and Tendencies'. **Factor de impacto en 2008: 0.488**
- Lucas, F.J., Molina, F. and Toval, A (2009). A Systematic Review of UML Model Consistency Management. Information and Software Technology, Vol. 51, Issue 12, pp. 1631-1645. **Factor de impacto en 2008: 1.2**

Aunque la normativa no obliga a ello, en este caso se ha preferido elaborar y añadir a estos artículos una documentación más extensa en la que se resumen los objetivos abordados en esta tesis, las aportaciones realizadas y los resultados alcanzados con el propósito de facilitar tanto a revisores, miembros del tribunal y a cualquier otro lector interesado la comprensión del trabajo llevado a cabo. La organización de esta documentación se puede consultar en la Sección 1.4.

Índice general

Índice de figuras

Índice de tablas

Resumen

A pesar de los numerosos avances realizados en el ámbito del desarrollo de software, la realidad es que actualmente el porcentaje de proyectos software que se completan de forma satisfactoria está en torno al 30 %. Por el contrario, el resto de proyectos abordados presentan problemas como retrasos en su planificación, fallos a la hora de satisfacer las necesidades de sus usuarios, son acabados con errores, exceden sus presupuestos o, lo que es peor, nunca son completados. Diferentes estudios revelan que un alto porcentaje de estos problemas se producen como consecuencia de una gestión inadecuada de las fases iniciales de desarrollo del proyecto como las relacionadas con la gestión de requisitos o el modelado conceptual del sistema. Recientemente, el paradigma de desarrollo dirigido por modelos ha emergido como una nueva área de la Ingeniería del Software en la que se propone que los modelos sean los artefactos clave que dirijan todo el proceso de desarrollo. Estos enfoques han introducido una nueva perspectiva para tratar con los requisitos que está principalmente basada en el uso de metamodelos. Además, en estas propuestas de desarrollo dirigido por modelos la calidad de los modelos conceptuales ha incrementado su importancia ya que los beneficios que pueden aportar dependen significativamente de las bondades de los modelos a partir de los que el software es construido. En esta tesis doctoral se presenta un método compuesto por un conjunto de técnicas alineadas con las prácticas propuestas por el paradigma de desarrollo dirigido por modelos y que se centra en reforzar la consideración de los requisitos del sistema y en mejorar la calidad de los modelos usados en el desarrollo como medio para contribuir a la mejora de la calidad del software obtenido a partir de ellos. La Ingeniería Web ha sido la disciplina elegida para ilustrar estas contribuciones debido al auge del tipo de sistemas de los que se ocupa y a que éstos se han convertido en sistemas críticos para las estrategias de negocio de muchas organizaciones.

Palabras clave: Ingeniería de Requisitos, Desarrollo Dirigido por Modelos, Ingeniería Web, metamodelado, usabilidad, seguridad, análisis de modelos.

Abstract

In spite of the plethora of advances made within the scope of software development, in reality only about 30% of software development projects are successfully completed. The remaining projects, meanwhile, suffer from problems such as schedule delays, failure to meet their stakeholders' needs, overrun their budget, are completed with errors or, in the worst case, are never completed. Various studies reveal that a high percentage of these problems are caused by inadequate management during the early stages of the development of the project such as those related to the requirements management or conceptual modelling of the system. A new software engineering area named Model Driven Engineering has recently emerged in which models have become the cornerstone of the development process. These approaches have introduced a new perspective for dealing with requirements which is mainly based on the use of requirements metamodels. Moreover, the quality of the conceptual models has increased in importance since the benefits of the model driven approaches significantly depend on how good the models from which the software is built are. In this PhD, we develop a new method composed of a set of techniques aligned to model driven development practices which focuses on the reinforcement of the consideration of requirements and the improvement of the quality of the models used in software development as a means of contributing towards improving the quality of the software obtained from them. The scope of Web Engineering has been chosen to illustrate this since its systems have become critical for the business strategies of many organizations.

This PhD meets the requirements to obtain the European Doctorate Mention and it has been presented in the form of *summary of publications* according to the authorization obtained from the University of Murcia with this aim.

Keywords: Requirements Engineering, Model Driven Engineering, Web Engineering, metamodelling, usability, security, model analysis.

Agradecimientos

Me gustaría aprovechar esta oportunidad para mostrar mi agradecimiento a todas las personas que a nivel profesional o personal han contribuido, de una u otra forma, a que este trabajo haya salido adelante.

En primer lugar debo empezar por mi familia. Este trabajo ha consumido mucho tiempo sin entender demasiado acerca de horarios o festivos, lo que en ocasiones obliga a dedicar a algunas cosas y personas menos tiempo del que uno debería. Muchas veces habéis tenido que estar allí donde yo debería haber estado para que pudiera seguir adelante con esto. Intentaré que en el futuro sea diferente.

Durante estos años han sido muchas las personas a las que he conocido y con las que he tenido la oportunidad de trabajar. A todos ellos les doy las gracias por las experiencias vividas así como a los revisores y miembros del tribunal por el tiempo dedicado a la revisión de este trabajo.

A Ambrosio Toval debo agradecerle la oportunidad de incorporarme al Grupo de Ingeniería del Software, sus valiosos consejos y el tiempo que ha dedicado a revisar los trabajos de esta tesis.

Las ocasiones en las que he estado fuera de Murcia siempre han supuesto experiencias muy enriquecedoras a muchos niveles. Mis meses de trabajo en Alicante fueron muy productivos y eso tengo que agradecérselo a Cristina Cachero. Trabajar tanto con Cris como con Jesús ha sido una grata experiencia que esperamos repetir en el futuro. También debo mostrar mi gratitud a Bashar Nuseibeh y a todas las personas con las que coincidí en *The Open University* sobre todo por enseñarme otras formas de trabajo y darme consejos muy útiles, especialmente a largo plazo. En este punto también me gustaría destacar a Carlos Blanco, con el que compartí parte de la estancia.

En el DIS y en el Grupo de Ingeniería del Software han sido unas cuantas las personas con las que he trabajado, desde José Luís Fernández, que hace ya algunos años me ayudó con Maude mientras desarrollaba mi PFC, hasta Jesús García-Molina con el que compartí un par de trabajos en la fase final de esta tesis pasando por Begoña Moros y Joaquín Nicolás con los que he compartido unas cuantas horas de charlas y reuniones sobre temas muy diversos y que siempre han estado ahí para lo que haya podido necesitar. Muchas gracias a todos y suerte en vuestros proyectos futuros.

Como no puede ser de otra forma, toda la gente que ha pasado por *la sala* tiene un espacio destacado porque se han convertido en mucho más que compañeros de trabajo.

Con Joaquín Lasheras y Francisco J. Lucas ha sido con los que he compartido más horas de viajes y trabajo. Parecía que no acabaríamos nunca la tesis pero al final acaba saliendo, así que os animo a que le deis el último empujón. Tanto a vosotros como a los que pasáis por allí de vez en cuando como Astrid y Javi Bermúdez y los que estáis allí muchas horas al día como Javi Cánovas, Javier Espinazo, Jesús y Óscar, deciros que ha sido un placer "trabajar" con todos vosotros.

No me gustaría olvidar a nadie pero seguro que lo he hecho así que sirva este párrafo final para dar las gracias a todos aquellos que alguna vez se interesaron por el estado de este trabajo o me dieron ánimos para seguir adelante.

Extended Abstract[1]

Introduction

Large quantities of software development projects are currently unsuccessfully completed. Recent studies such as that of [3] carried out over thousands of software projects reveal that only 32 % of projects are successfully completed, that is, are delivered on time, within budget and with the required features and functionalities. On the contrary, 44 % of the projects overrun their time and money budgets and/or are delivered with less than the required functionality. What is worse, 24 % of projects completely fail, which means that are cancelled prior to completion or delivered and never used.

When these studies analyse the causes of these problems in greater depth, it is recurrently detected that one of the main reasons for these failure percentages lies in an inadequate management of the early stages of project development such as those related to requirements management and conceptual modelling of the system [3, 4]. Furthermore, the detected problems are not specific to the software developed for concrete domains but appear in all kinds of software (industrial, e-commerce applications, etc.).

The Model Driven Engineering (MDE) [5, 6] has recently emerged as a new area in Software Engineering that provides a cost-effective, reliable and rapid application development through which to obtain products faster and with more quality. In these approaches, models have become the cornerstone of the development process which is completely driven by models with different abstraction levels and the transformations among them.

These model driven approaches are introducing a new perspective with which to deal with requirements. Since the entire development process in MDE is driven by models, the requirements should also be represented as models through the use of requirements metamodels that formally define the concepts and relationships involved [7]. These metamodels are more suitable for integration into an MDE process than traditional techniques such as use cases. With regard to conceptual models, their quality has always been important since it influences the quality of the final system developed from them. However, the quality of these models is even more important in model driven proposals since their

[1] This chapter offers a summary of the aim and contributions of this PhD thesis and it has been included as part of the requirements for the obtention of the European Mention for this PhD

benefits significantly depend on how good the models on which they are based will be [6]. Furthermore, the use of multiple models or views for a system may cause consistency problems among models which must also be considered [8, 9, 10]. Thus, it is critical to provide modellers with techniques for verifying their models, detecting errors in each one or even inconsistencies among them as a means of obtaining better models from which to build the final system [6, 11].

In view of this situation, this PhD attempts to mitigate the aforementioned problems through a set of techniques aimed at reinforcing the consideration of requirements and improving the quality of the conceptual models as a means to increase the quality of the systems finally developed from them. Owing to the impact and benefits offered by the MDE approaches, the contributions proposed will be aligned to model driven practices. Although most of them can be used in any scope, the Web Engineering discipline [12], which focuses on the development, deployment and maintenance of Web-based systems, has been chosen to apply these contributions.

Web Engineering has been chosen as field of work for several reasons. The exponential growth of the Internet in the last decade has motivated the presence of web-based systems in numerous domains used by millions of users around the world. The complexity of these systems has increased and they have become critical systems for the business strategies of many organizations [13]. However, the development of this kind of systems is not exempt from errors. Studies such as [14, 15, 16] highlight that the top five problem areas of large-scale WE projects are (1) failure to meet business needs (84 %), (2) project schedule delays (79 %), (3) budget overrun (63 %), (4) lack of required functionality (53 %) and (5) poor quality of deliverables (52 %). In fact, these problems are quite similar to those mentioned above and are again a symptom of an inadequate management of the tasks related to the requirements workflow of the project. Thus, the consideration of requirements in the development of this kind of projects must be reinforced [1, 17].

The proposals for Web-based systems development are also being aligned to MDE within an area denominated as Model Driven Web Engineering [18] which proposes their development through a process based on the use of several models focused on the different views of the system such as the navigation of the users or the presentation details.

Within the scope of Web based systems there are some requirements related to attributes such as usability or security that require from a more detailed attention [14, 19]. Studies such as [19, 20] in the case of usability or [14, 21] with regard to security advise that these attributes must also be considered from early stages of development as requirements management and modelling since, again, this provides important benefits over to the quality of the system finally developed.

From our point of view, the alignment to the model driven approach and the importance of the requirements management and modelling phases make this discipline suitable for the integration and validation of the techniques proposed in this PhD.

Aims and contributions

As it was previously mentioned, the global goal of this PhD is the design of a method that integrates a set of techniques aimed at the improvement of software quality from the early stages of development, specifically, those related to requirements management and conceptual modelling. This general goal can be divided into more concrete sub-goals and tasks for their achievement, which are detailed below.

With regard to the part of the method related to requirements management, the main aim is to provide techniques that allow more emphasis to be placed upon the adequate elicitation and management of requirements and which complement those model driven methodologies that do not consider requirements in a specific way. Specific extensions for the elicitation and management of both usability and security requirements will also be provided.

With regard to the part of the method related to conceptual modelling, the main aim is to provide a mechanism for the improvement of the quality of the models used in the system development as a means to improve the quality of the software built from them. This will be based on the use of formal techniques for the verification, validation and consistency analysis of the models involved.

A set of contributions has been carried out in order to achieve these aims:

- A requirements metamodel that formalizes the concepts involved in the requirements elicitation process and the relationships among them has been proposed for the reinforcement of Requirements Engineering related tasks, which follows the trend introduced by MDE approaches [22, 23].

- The aforementioned requirements metamodel has been integrated with a measurement metamodel with the aim of offering support for linking the elicited requirements with measures that can assist in checking their fulfilment in the developed system [24, 25, 26]. This proposal has been integrated into the scope of Web Engineering methodologies [27] and a notation for it based on the use of a UML profile has been provided.

- A proposal for the consideration of usability requirements from the early stages of system development has been designed. In contrast to the majority of the existing approaches, our proposal benefits from the advantages of model based usability evaluation rather than delaying this task until the system has been completely developed as usually occurs [22, 28].

- An initial proposal for the consideration of security requirements from early stages and their introduction in a model driven development process have been carried out [29, 30].

- A set of techniques have been developed for the analysis of conceptual modelling. In our case, these techniques are based on the use of formal methods and their

usefulness has been proved in the scope of Web Engineering. Specifically, techniques for the verification of models have been developed and presented in [31, 32]. Furthermore a systematic literature review aimed at analysing the state-of-the-art with regard to model consistency in software development has been carried out [8] and has been used as a basis for the development of an initial proposal for consistency analysis in the scope of Web Engineering [33].

PhD frame and publications

This section shows the projects in which this PhD has been carried out and the publications achieved.

Research group and research stays

This PhD has been mainly developed in the Software Engineering Research Group at the University of Murcia. The work has additionally been complemented with two periods of research at other universities. One of these took place in the Software Engineering and Design Group at The Open University (Milton Keynes, UK), under the supervision of Dr. Bashar Nuseibeh and the other one took place in the Web Engineering and Data Warehouses Research Group at the University of Alicante under the supervision of Dra. Cristina Cachero.

This PhD has been partially financed by the Fundación Séneca (Región de Murcia) and is mainly framed within the DEDALO (Development of Quality Systems based on models and requirements, TIN2006-15175-C05-03, developed between the years 2007 and 2009) and PANGEA (Process for globAl requiremeNts enGinEering and quAlity, TIN2009-13718-C02-02, that will be developed in the period 2010-2012) projects financed by the Spanish Ministry of Science and Technology, along with some thematic networks related to Model Driven Engineering (TIN2005-25866-E, 2006-2009), Software Quality (TIN2005-24055-E, 2005-2008) and the MAUDE language (TIN2006-26882-E, 2006-2009) financed by the same Ministry.

Publications

Parts of the results of this work have been presented and discussed in various peer-review forums. The publications in which the author has been involved are listed below.

Journals indexed in the ISI Journal Citation Reports (JCR)

- Molina, F. and Toval, A (2009). Integrating usability requirements that can be evaluated in design time into Model Driven Engineering of Web Information Systems. Journal of Advances in Engineering Software, Vol. 40, Issue 12, pp. 1306-1317. **JCR Index in 2008: 1.188**

- Molina, F., Pardillo, J., Cachero, C. and Toval, A (2010). An MDE Modelling Framework for Measurable Goal-oriented Requirements. International Journal of Intelligent Systems. Accepted for publication in the Special Issue on Goal-Driven Requirements Engineering. **JCR Index in 2008: 0.860**

- Lucas, F.J., Molina, F. and Toval, A (2009). A Systematic Review of UML Model Consistency Management. Information and Software Technology, Vol. 51, Issue 12, pp. 1631-1645. **JCR Index in 2008: 1.2**

- Sánchez, O, Molina, F., García-Molina, J. and Toval, A (2009). ModelSec: a generative architecture for model-driven security. Accepted for publication in the Special Issue 'Security in Information Systems: New Advances and Tendencies' of the Journal of Universal Computer Science. **JCR Index in 2008: 0.488**

Articles published in LNCS Proceedings

- Molina, F., Pardillo, J., Toval, A. Modelling Web-based Systems Requirements using WRM. 2nd Int. Workshop on Web Usability and Accessibility (IWWUA'08) in conjunction with the 9th Int. Conf. on Web Information Systems Engineering (WISE'08). LNCS Volume 5176, pp. 122-131. Auckland, New Zealand, 2008

- Pardillo, J., Molina, F., Cachero, C., Toval, A. A UML Profile for Modelling Measurable Requirements. 4th Int. Workshop on Foundations and Practices of UML (FPUML'08), in conjunction with the 27th International Conference on Conceptual Modelling, ER 2008. LNCS 5232, pp. 123-132. Barcelona, 2008

- Molina, F., Toval, A. A generic approach to improve navigational model usability based upon requirements and metrics. 1st Int. Workshop on Web Usability and Accessibility (IWWUA'07) in conjunction with the 8th Int. Conf. on Web Information Systems Engineering (WISE'07). LNCS Volume 4832, pp. 511-516. Nancy, France, 2007

Chapters in International Books

- Molina F., Lucas, F.J., Toval, A., Vara, J.M., Marcos, E. Towards Quality Web Information Systems through Precise Model Driven Development. Chapter in Handbook of Research on Web Information Systems, pp. 317-355. Idea Group Publishing, 2008. ISBN: 978-1-59904-847-5

International Conferences/Workshops

In addition to the work published in LNCS proceedings shown above, other publications are:

- Lucas, F. J., Molina, F., Toval, A. A Rule-based Approach for Managing Semantic Inconsistencies in Web Information Systems Development. 5th International Workshop on Automated Specification and Verification of Web Systems (WWV'09). Austria, 2009
- Sánchez, O., Molina, F., García-Molina, J., Toval, A. A model driven approach for generating code from security requirements. International Workshop on Security In Information Systems (WOSIS'09), pp, 119-126. Milan (Italy), 2009
- Molina, F., Cachero, C., Pardillo, J., Toval, A. Towards a Requirements-Aware Common Web Engineering Metamodel. Latinoamerican Conference of Web Engineering (LAWEB'08), pp. 75-82. IEEE Digital Library. Brasil, 2008.
- Lucas, F.J., Molina F., Toval, A., De Castro, M.V., Cáceres, P., Marcos, E. Precise WIS Development. International Conference on Web Engineering (ICWE '06), pp. 71-76. Menlo Park, California (USA). ACM Proceedings. ISBN: 1-59593-352-2.

Chapters in Spanish Books

- Molina, F., Cachero, C., Pardillo, J., Toval, A. Metamodelado de requisitos medibles. In Quality of Software Products and Processes, Chapter 20. Editors: C. Calero and M. A. Moraga. Rama Publishing. Accepted to be published in 2010.

Spanish and Iberoamerican Conferences

- Molina, F., Cachero, C., Pardillo, J., Toval, A. Metamodelo y perfil UML para el modelado orientado a metas de requisitos medibles. 13th Conference on Software Engineering and Databases (JISBD'08), pp. 345-356. Gijón, Spain. 7-10 October. Acceptance Rate: 30 %
- Molina, F., Toval, A. Una propuesta para la evaluación temprana de usabilidad en Sistemas de Información Web. IX Congreso Internacional de Interacción Persona-Ordenador (Interaction'08), pp. 243-252. Albacete, Spain. 9-11, 2008.
- Molina F., Lucas, F.J., Toval, A., Vara, J.M., Marcos, E.. Soporte CASE para el Desarrollo Preciso de Sistemas de Información Web. Conferencia Ibero-Americana IADIS WWW/Internet 2006, pp. 49-56. Murcia (España). IADIS Press. ISBN: 972-8924-20-8

Capítulo 1

Introducción

1.1. Introducción

A pesar de los múltiples avances realizados en el ámbito del desarrollo de software, la realidad es que, actualmente, todavía son muchos los proyectos de desarrollo que no finalizan de forma satisfactoria [4, 34, 35]. Estudios recientes realizados sobre miles de proyectos software revelan que tan sólo el 32 % de los mismos son completados satisfactoriamente, es decir, son entregados en los plazos previstos, dentro del presupuesto estimado y tienen las características y funcionalidades requeridas [3]. Por el contrario, el 44 % de los proyectos analizados superan sus estimaciones en cuanto a tiempo y coste y/o son entregados sin disponer de las funcionalidades solicitadas. Lo que es aún peor, el 24 % de los proyectos software abordados acaban dándose por fallidos, es decir, son cancelados antes de finalizar o son entregados pero nunca son usados.

Cuando estos estudios profundizaron en las causas que motivaban estos porcentajes de fracaso se detectó que muchos de los problemas eran consecuencia de una gestión inadecuada de las fases iniciales de desarrollo del proyecto, como son aquellas relacionadas con la captura de requisitos o el modelado conceptual del sistema a desarrollar. Además, los problemas detectados no afectaban al software desarrollado en dominios concretos, sino que eran problemas generales que, en mayor o menor medida, aparecían en multitud de ámbitos (ingenieriles, industriales, comercio electrónico, etc.).

Por otro lado, las propuestas de desarrollo de software dirigido por modelos (*Model Driven Engineering*, MDE [5, 6]) han emergido recientemente como un nuevo área de la Ingeniería del Software que proporciona un marco de desarrollo fiable y efectivo en coste que permite obtener productos software de forma más rápida y con mayor calidad [36]. Los modelos son la piedra angular sobre la que se basan estas propuestas ya que se propone que todo el proceso de desarrollo sea dirigido por modelos con diferentes niveles de abstracción y transformaciones entre ellos. De esta forma, los modelos aumentan su importancia ya que éstos habían sido tradicionalmente usados como un mecanismo de documentación y comunicación mientras que en MDE se convierten en los artefactos software clave que dirigen todo el proceso de desarrollo.

Los enfoques MDE han introducido una nueva perspectiva para tratar con los requisitos del sistema. Debido a que en MDE todo el proceso de desarrollo es dirigido por modelos, los requisitos deberían también ser representados como modelos mediante el uso de metamodelos de requisitos que definen formalmente los conceptos implicados en la elicitación de requisitos y las relaciones entre ellos [7]. Estos metamodelos permiten una integración de los requisitos en el ciclo de desarrollo propuesto por MDE más adecuada que la ofrecida por otras técnicas tradicionales como los casos de uso.

Con respecto a los modelos conceptuales del sistema, su calidad siempre ha sido importante en el desarrollo de software ya que ésta incide en la calidad del sistema obtenido finalmente a partir de ellos. Sin embargo, la calidad de estos modelos cobra si cabe más importancia en las propuestas MDE ya que sus beneficios dependen en gran medida de la calidad de los modelos sobre los que se basan [6]. Además, el uso de diferentes modelos para un mismo sistema puede introducir problemas de consistencia entre los mismos que también deben ser considerados [8, 9, 10]. Por tanto, resulta crítico ofrecer a los modeladores técnicas que permitan verificar que los modelos son correctos (al menos, con respecto a un conjunto de propiedades determinado), detecten errores en cada modelo o inconsistencias entre ellos y, en definitiva, permitan obtener mejores modelos a partir de los que construir el sistema final [6, 11].

En vista de los porcentajes de fracaso mencionados así como del impacto y beneficios ofrecidos por el enfoque MDE, esta tesis doctoral intentará ayudar a mitigar estos problemas ofreciendo un conjunto de contribuciones orientadas a reforzar la consideración de los requisitos y la mejora de la calidad de los modelos conceptuales, como medio para contribuir a mejorar la calidad de los sistemas desarrollados a partir de ellos. Las contribuciones realizadas estarán alineadas con las prácticas propuestas por el paradigma MDE y se integrarán en un método orientado a mejorar los artefactos obtenidos en las fases iniciales de desarrollo de proyectos software, como los modelos de requisitos y los modelos conceptuales del sistema.

Para la aplicación y validación de estas contribuciones se ha seleccionado el ámbito de la Ingeniería Web aunque, tal y como se irá detallando en los siguientes capítulos, la mayoría de ellas podrían ser aplicadas en cualquier otro dominio. La Ingeniería Web se centra en la aplicación de principios ingenieriles para el desarrollo, despliegue y mantenimiento de sistemas web [12] (también denominados Sistemas de Información Web, SIW).

La elección de la Ingeniería Web como ámbito de trabajo ha estado motivada por diferentes razones. El crecimiento exponencial de Internet en la última década ha hecho que los sistemas web estén presentes en multitud de ámbitos, siendo utilizados por millones de usuarios cada día. La complejidad de estos sistemas está en continuo aumento y éstos se han convertido en sistemas críticos para las estrategias de negocio de muchas organizaciones [13].

Sin embargo, el desarrollo de este tipo de sistemas tampoco está exento de errores. Diferentes estudios destacan este hecho [14, 15, 16], siendo las principales áreas de problema los fallos a la hora de satisfacer las necesidades para las que los sistemas estaban destinados, los retrasos sobre la planificación del proyecto, la necesidad de incrementar el

presupuesto estimado, la ausencia de funcionalidad requerida y la calidad deficiente de los entregables.

Estos problemas son similares a los encontrados en el desarrollo de otros sistemas software siendo de nuevo indicativos de una gestión inadecuada de las fases iniciales de gestión del proyecto. Estudios como los realizados por Escalona *et al.* [1, 37] concluyen que la consideración de requisitos es un área a ser reforzada en Ingeniería Web ya que las metodologías existentes están principalmente centradas en aspectos como el diseño del sistema, dejando de lado el tratamiento de los requisitos que, en el mejor de los casos, es considerado de forma tangencial. Las propias organizaciones dedicadas al desarrollo de sistemas basados en web ratifican la importancia de una gestión adecuada de los requisitos. Un estudio realizado por Lang y Fitzgerald [17] sobre 160 organizaciones dedicadas al desarrollo de estos sistemas indicó que el 60 % de ellas consideraba que uno de los principales problemas en el desarrollo de sus proyectos era la claridad y estabilidad de los requisitos.

Por otro lado, las propuestas de desarrollo de sistemas web están siendo alineadas con MDE dando lugar a un área de trabajo denominada Ingeniería Web *dirigida por modelos* (*Model Driven Web Engineering*, MDWE [18]). Siguiendo la filosofía MDE, en MDWE se propone que el desarrollo del sistema sea realizado mediante el uso de diferentes modelos centrados en cada una de las vistas del sistema como, por ejemplo, la navegación de los usuarios o los detalles de presentación.

Además, en el desarrollo de sistemas web existen determinados requisitos que requieren de una atención tanto o incluso más detallada que en el caso de otros tipos de software como, por ejemplo, los requisitos relacionados con la usabilidad o la seguridad del sistema. La usabilidad es un aspecto crítico para el éxito de los sistemas web y, como diferentes estudios corroboran [19, 20], debe ser considerada desde las fases iniciales de desarrollo del proyecto, como la elicitación de requisitos y el modelado conceptual. Algo similar ocurre con la seguridad. La propia naturaleza de estos sistemas y su funcionamiento a través de Internet les hace estar más expuestos a problemas de seguridad siendo más susceptibles a vulnerabilidades que las aplicaciones tradicionales. Por ello, la seguridad se convierte en otro aspecto crítico y, de nuevo, se recomienda su consideración desde fases iniciales de desarrollo como mejor estrategia para la construcción de sistemas web seguros [21, 38].

Los problemas que los proyectos de Ingeniería Web presentan relacionados con la gestión inadecuada de los requisitos, el alineamiento con MDE y la existencia de atributos como usabilidad o seguridad que requieren de una consideración adecuada desde fases iniciales del proyecto configuran, desde nuestro punto de vista, un marco adecuado en el que introducir las contribuciones propuestas en esta tesis doctoral.

1.2. Hipótesis y objetivos

La hipótesis de partida sobre la que se trabajará en esta tesis doctoral es que resulta factible definir contribuciones que permitan incidir en el tratamiento dado a las fases

iniciales de desarrollo de proyectos software, intentando que una mayor consideración de estas etapas y la mejora de los artefactos creados en ellas redunde en la calidad de los sistemas desarrollados finalmente. Utilizando esta hipótesis como punto de partida, se define el siguiente objetivo global para la tesis:

- **Objetivo:** diseñar un método integrado por un conjunto de contribuciones alineadas con las prácticas propuestas por los enfoques MDE destinadas a reforzar las fases iniciales de desarrollo de sistemas software y mejorar los artefactos creados en ellas, como los modelos de requisitos y modelos conceptuales que posteriormente serán usados para construir el sistema.

Este objetivo general se puede concretar a través de una serie de objetivos parciales, los cuáles se detallan a continuación:

- **Objetivo 1:** analizar la incidencia que las fases relacionadas con la elicitación de requisitos y el modelado conceptual del sistema tienen en el éxito final de proyectos de desarrollo software. En este análisis se incluirá el estudio de los paradigmas de desarrollo dirigido por modelos, analizando tanto las características como las técnicas y herramientas promovidas por estas propuestas.
- **Objetivo 2:** desarrollar un conjunto de contribuciones alineadas con el paradigma MDE que, integradas en un método de desarrollo, contribuyan a mitigar las carencias detectadas durante el análisis del estado del arte. Este objetivo se descompone en los siguientes subobjetivos:
  - Introducción de técnicas que permitan reforzar la atención recibida por los requisitos durante el desarrollo de software. Este subobjetivo será alcanzado mediante el desarrollo de un metamodelo de requisitos que formaliza los conceptos implicados en el proceso de elicitación de requisitos y las relaciones entre ellos. A su vez, este metamodelo servirá además como base para la realización de las siguientes contribuciones.
  - Integración de técnicas para la elicitación de requisitos con técnicas para la medición de software como medio para ligar requisitos y medidas de su cumplimiento, ofreciendo así soporte para la elicitación de requisitos cuantificables.
  - Definición de estrategias basadas en el uso de metamodelos y transformaciones que permitan el modelado de requisitos relacionados con la usabilidad y seguridad del software desde fases tempranas de su desarrollo.
  - Definición de actividades para la verificación precisa y análisis de modelos. Estas actividades harán uso de técnicas formales y complementarán la fase de modelado conceptual del sistema con el propósito de mejorar los modelos creados durante esta etapa.
- **Objetivo 3:** diseñar e implementar los prototipos software que den soporte a las contribuciones comentadas en el Objetivo 2.

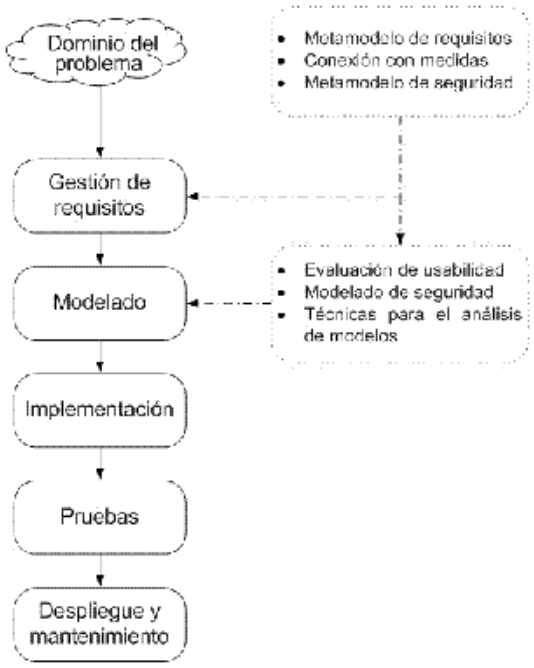

Figura 1.1: Integración de contribuciones en un ciclo clásico de desarrollo de software

- **Objetivo 4:** mostrar la aplicación de las contribuciones obtenidas para la consecución del Objetivo 2 utilizando como campo de estudio la disciplina de Ingeniería Web.

La Figura 1.1 muestra una visión general acerca de cómo las contribuciones que se acaban de mencionar se integrarían en un ciclo clásico de desarrollo de software. Como se definió en la formulación de la hipótesis de partida, las contribuciones desarrolladas reforzarán las etapas iniciales del desarrollo como el análisis de requisitos y modelado conceptual. En el Capítulo 3 se explicará en detalle la forma en la que estas contribuciones son integradas en el ciclo de vida.

1.3. Marco de la tesis

A continuación se detalla el entorno de trabajo en el que se ha desarrollado la presente tesis doctoral, resaltándose tanto los Grupos de Investigación en los que se ha desarrollado como los proyectos de I+D en los que se ha enmarcado.

1.3.1. Grupos de investigación y estancias

La tesis ha sido desarrollada principalmente en el Grupo de Investigación en Ingeniería del Software de la Universidad de Murcia (www.um.es/giisw) y complementada con estancias de investigación en el Grupo de Ingeniería Web y Almacenes de Datos de la

Universidad de Alicante, bajo la supervisión de la Dra. Cristina Cachero y en el *Software Engineering and Design Group* perteneciente a The Open University (Milton Keynes, Reino Unido), bajo la supervisión del Dr. Bashar Nuseibeh.

El trabajo realizado ha sido financiado por la Fundación Séneca (Región de Murcia) y diferentes proyectos de investigación que son comentados en la siguiente sección.

1.3.2. Acciones de I+D+I

1.3.2.1. Proyectos de investigación aplicada y transferencia tecnológica

Los proyectos de investigación en los que se enmarcó este trabajo han sido los siguientes:

- PANGEA (Process for globAl requiremeNts enGinEering and quAlity, TIN2009-13718-C02-02), integrado en el proyecto PEGASO (Processes for the Improvement of Global Software Development) en el que participan la Universidad de Castilla-La Mancha y la Universidad de Murcia. El proyecto está financiado por el Ministerio de Ciencia y Tecnología comenzando a finales de 2009 y acabando su desarrollo en el año 2012 y ha financiado la parte final de esta tesis.
- DEDALO (Desarrollo de sistEmas de caliDad bAsado en modeLos y requisitOs, TIN 2006-15175-C05-03). Integrado en el proyecto META (Models, Environments, Transformations and Applications), un proyecto coordinado entre cuatro universidades españolas (Universidad Politécnica de Valencia, Universidad de Castilla-La Mancha, Universidad Politécnica de Cartagena y Universidad de Murcia) y el European Software Institute (ESI). El proyecto fue financiado por el Ministerio de Ciencia y Tecnología, comenzando en 2006 y acabando su desarrollo a finales de 2009.
- MELISA-GREIS (Metodología para el desarrollo global del Software, PAC08-0142-335) en el que participan las Universidades de Castilla-La Mancha y Murcia. El proyecto está financiado por la Consejería de Educación y Ciencia, en el marco del Plan Regional de Investigación Científica, Desarrollo Tecnológico e Innovación de Castilla-La Mancha, siendo desarrollado entre los años 2008 y 2010.
- GARTIC (Gestión Automatizada de Requisitos basada en Reutilización para PYMES del sector TIC), un proyecto de **transferencia tecnológica** firmado por el Grupo de Investigación en Ingeniería del Software de la Universidad de Murcia y el Centro Tecnológico de las Tecnologías de la Información y las Comunicaciones (CENTIC) de la Región de Murcia, que contó con la participación de cinco empresas regionales a las que se formó para la utilización de técnicas de Ingeniería de Requisitos en sus proyectos de desarrollo.
- FoMDAs (Formalización de (Meta-) Modelos y transformaciones en un marco MDA para el Desarrollo Automático de SIW, URJC-CM-2006-CET-038), financiado por la Universidad Rey Juan Carlos y la Comunidad de Madrid. Este proyecto se desarrolló entre los años 2006 y 2008.

- PRESSURE (PREcise Software modelS and reqUirements REuse, TIC2003-07804-C05-05), integrado en el Proyecto DYNAMICA (DYNamic and Aspect-Oriented Modeling for Integrated Component-based Architectures). En el proyecto participaron cinco universidades españolas (Universidad Politécnica de Valencia, Universidad de Castilla-La Mancha, Universidad Carlos III, Universidad Politécnica de Cartagena y Universidad de Murcia) y fue financiado por el Ministerio de Ciencia y Tecnología entre los años 2003 y 2006.

1.3.2.2. Redes temáticas

El doctorando también ha formado parte de varias redes temáticas relacionadas con el contenido de la tesis doctoral. Los detalles de estas acciones se muestran a continuación:

- Red CALIPSO (Calidad de Producto y Proceso Software, TIN2005-24055-E), en la que participan 10 universidades españolas, 5 universidades iberoamericanas y cinco empresas, con un total de más de 100 investigadores y tres años de duración. El objetivo principal de esta red fue servir como punto de encuentro de todos ellos para trabajar en la mejora de la calidad de los productos y procesos software. La red, dirigida por la Dra. Coral Calero (Universidad de Castilla-La Mancha), se desarrolló entre los años 2005 y 2007 y posteriormente se extendió durante el año 2008.

- Red DSDM (Desarrollo de Software Dirigido por Modelos, TIN2005-25866-E), formada por más de 100 investigadores de 15 universidades españolas y 4 empresas. Esta red sirve como punto de encuentro a todos los grupos de trabajo nacionales que trabajan en el ámbito del desarrollo de software dirigido por modelos. La red esta dirigida por el Dr. Antonio Vallecillo (Universidad de Málaga) e inicialmente se desarrolló entre los años 2006 y 2008 para ser posteriormente extendida hasta finales del año 2009.

- Red Maude (TIN2006-26882-E), formada por más de 30 investigadores de diferentes universidades españolas y que se centra en el estudio y aplicaciones del lenguaje formal Maude en el ámbito del desarrollo de software. La red se desarrolló entre los años 2006 y 2008 y posteriormente se extendió hasta el año 2009 bajo la dirección del Dr. Narciso Martí-Oliet (Universidad Complutense de Madrid).

- Red RETISTRUST (Red temática de Investigación en el campo de la Seguridad y confianza para los Sistemas de Información en una Sociedad Conectada, TIN2006-26885-E), en la que participaron diferentes universidades españolas e iberoamericanas, así como la Excelentísima Diputación de Ciudad Real. Su objetivo principal fue consolidar, unificar y divulgar los conocimientos sobre la seguridad y asentar la confianza en la tecnología. Se desarrolló en el año 2008 bajo la dirección del Dr. Eduardo Fernández-Medina (Universidad de Castilla-La Mancha).

1.4. Organización de la tesis

Este documento se ha organizado en los siguientes capítulos:

- En el Capítulo 1 se ha mostrado una visión general acerca de los objetivos de esta tesis y su marco de desarrollo.
- En el Capítulo 2 se mostrará el estado del arte en aquellas disciplinas relacionadas con esta tesis doctoral como la Ingeniería de Requisitos, el Desarrollo de Software Dirigido por Modelos o la Ingeniería Web.
- En el Capítulo 3 se mostrará el método que integra las contribuciones propuestas en esta tesis, explicándose las fases y actividades que lo componen.
- El Capítulo 4 detallará las contribuciones que esta tesis realiza en el área de Ingeniería de Requisitos.
- En el Capítulo 5 se explicará la propuesta presentada para la evaluación temprana de usabilidad sobre los modelos conceptuales usados por las metodologías de desarrollo en el campo de la Ingeniería Web.
- El Capítulo 6 mostrará las estrategias propuestas para el análisis de los modelos conceptuales usados en el desarrollo de software, usando los modelos de Ingeniería Web como campo de estudio.
- En el Capítulo 7 se analizarán los objetivos alcanzados junto con las conclusiones, el contraste de resultados obtenidos y las líneas de trabajo futuro.

Capítulo 2

Estado del arte

2.1. Introducción

En este capítulo se muestra el estado del arte en las líneas de investigación abordadas, concretamente, la Ingeniería de Requisitos, el Desarrollo de Software Dirigido por Modelos y la Ingeniería Web, de la que se proporcionará una introducción por ser el ámbito de aplicación de las contribuciones realizadas. En la Sección 2.2 se ofrecerá una visión general acerca de los objetivos de la Ingeniería de Requisitos así como de la conexión entre este área y la de medición de software con el objetivo de asociar a los requisitos medidas para la evaluación de su cumplimiento. Uno de los paradigmas que más está influyendo en el desarrollo de software en los últimos años es el desarrollo de software dirigido por modelos cuyos fundamentos, ventajas y líneas de investigación abiertas serán comentados en la Sección 2.3.

En la Sección 2.4 se ofrecerá una visión general sobre Ingeniería Web y las diferentes metodologías surgidas en este ámbito, la cuáles se están alineando de forma reciente con el paradigma de desarrollo dirigido por modelos. El análisis de estas metodologías sirvió para detectar determinadas carencias a las que esta tesis doctoral también aportará contribuciones. La primera de estas contribuciones pretende reforzar el tratamiento que las metodologías de desarrollo alineadas con el desarrollo dirigido por modelos dan a los requisitos y será motivada en la sección 2.4.1. La segunda está relacionada con la consideración de requisitos relacionados con la usabilidad del software desde fases tempranas de desarrollo y se motivará en la Sección 2.4.2. Por último, la tercera contribución se centrará en la verificación y análisis preciso de los modelos utilizados para el desarrollo de software y será detallada en la sección 2.4.3.

2.2. Ingeniería de Requisitos

La Ingeniería de Requisitos (IR) es una de la áreas de la Ingeniería del Software considerada como crítica para el éxito de un proyecto de desarrollo software [39]. Existen

diferentes definiciones de IR pero, de forma general, la IR para sistemas software es entendida como un proceso dedicado a descubrir el propósito de un sistema, identificando a sus *stakeholders,* así como las necesidades de éstos, y a documentar todo ello en una forma adecuada para su análisis, comunicación y posterior implementación [40]. Se considerarán *stakeholders* todas aquellas personas u organizaciones que se verán afectadas por el sistema y que tienen influencia directa o indirecta sobre los requisitos de éste.

La gestión inadecuada de los requisitos es la causa probada de multitud de fallos y fracasos en proyectos software [3, 4, 34, 35]. Por el contrario, una gestión adecuada de los mismos ayuda a mejorar la calidad del software y la productividad de sus procesos de desarrollo [39]. Sin embargo, esta gestión adecuada de los requisitos ha sido considerada desde siempre como una de las actividades más complejas en el desarrollo de software [41] ya que debe enfrentarse a diferentes dificultades como que los *stakeholders* puedan ser numerosos y distribuidos, con diferentes objetivos que podrían ir variando y estar en conflicto, con objetivos difíciles de descubrir o, incluso, que se de el caso de que los *stakeholders* ni siquiera conozcan o sean capaces de transmitir sus propios requisitos [40].

Por ello, y aunque existían ya muchos esfuerzos previos en este área, fue a principios de la década de los noventa cuando se comenzó a prestar una mayor atención a la IR y ésta emergió como una nueva disciplina en la que aparecieron múltiples enfoques dedicados a la gestión adecuada de los requisitos del software [40]. Entre estos enfoques podemos destacar, por ejemplo, las propuestas orientadas a metas (*Goal Oriented Requirements Engineering*) [42], dirigidas por aspectos [43], dirigidas por puntos de vista [44, 45] o basadas en el uso de plantillas para los requisitos [46].

En los últimos años, el impacto de las propuestas de desarrollo de software dirigido por modelos, de las que se ofrecerá más detalle en la Sección 2.3, ha introducido una nueva perspectiva para tratar con los requisitos. En estas propuestas, todo el proceso de desarrollo de software es dirigido por modelos, así que los requisitos son también representados como modelos gracias al uso de metamodelos de requisitos [7]. De forma general, un metamodelo actúa como modelo de un lenguaje de modelado y define formalmente los conceptos y restricciones que permiten la creación de modelos válidos en ese lenguaje [47]. Por tanto, los metamodelos de requisitos deberían definir de forma precisa los conceptos y relaciones implicados en el proceso de Ingeniería de Requisitos [7].

Recientemente han aparecido diferentes propuestas para el metamodelado de requisitos pero, como se detallará en el Capítulo 4, éstas presentan una gran disparidad en aspectos como, por ejemplo, los conceptos considerados por cada una, lo que hace que no sea posible hablar de un metamodelo de referencia [7]. De hecho, el problema de qué conceptos deben tenerse en cuenta durante el proceso de IR y las definiciones precisas de éstos sigue abierto, existiendo esfuerzos recientes basados en el uso de ontologías que intentan contribuir a determinar estos conceptos [48].

En el Capítulo 4 de esta tesis mostraremos una comparativa entre las propuestas más relevantes que han surgido en los últimos años para el metamodelado de requisitos. Usando esta comparativa como base, presentaremos un metamodelo de requisitos propio que intenta unificar los conceptos más importantes detectados en las diferentes propuestas

estudiadas. Este metamodelo de requisitos será utilizado posteriormente con diferentes objetivos. Por un lado, tal y como comentaremos en la Sección 2.2.1, el metamodelo de requisitos se integrará con un metamodelo de medición de software con el objetivo de ofrecer soporte para la conexión entre los requisitos y medidas que ayuden comprobar su cumplimiento. Por otro lado, este metamodelo será extendido con conceptos específicos del ámbito de la seguridad, y formará parte de una arquitectura generativa diseñada para la consideración temprana de este tipo de requisitos siguiendo el enfoque denominado *seguridad dirigida por modelos*, que será explicado en la Sección 2.3.1.

2.2.1. Requisitos y medición

Durante el proceso de elicitación de requisitos de un sistema, éstos suelen ser especificados de una forma cualitativa. Algunos ejemplos de estos requisitos especificados cualitativamente podrían ser el requerir que los tiempos de respuesta del sistema sean rápidos o que la navegación por el sistema desarrollado sea amigable. Los requisitos especificados de esta forma suelen ser ambiguos y difíciles de verificar lo que, como destaca Glinz [49], puede conllevar una serie de problemas como que:

- Los desarrolladores construyan un sistema que no cumpla con las expectativas esperadas por sus *stakeholders*, hecho que provocará falta de satisfacción en los mismos o incluso, en determinados casos, que el sistema no sea útil y no se use.
- Los desarrolladores construyan un sistema que ofrezca a sus *stakeholders* más de lo que éstos necesitan, lo que muy probablemente habrá implicado la inversión de presupuestos y tiempos de desarrollo por encima de los necesarios.
- Los desarrolladores y clientes no estén de acuerdo acerca de si el sistema cumple con determinados requisitos y no existan criterios claros que ayuden a decidir quién tiene la razón.

Una de las soluciones tradicionalmente adoptadas para resolver estos problemas consiste en la cuantificación de requisitos, es decir, en la definición de métricas asociadas con los requisitos, dando así soporte al concepto de *requisitos medibles*. La idea de definir requisitos junto con métricas no es nueva, ya que los primeros trabajos en este sentido surgieron hace décadas [50, 51, 52] pero actualmente ésta sigue siendo un área de investigación activa [49, 53, 54].

Desde el punto de vista de esta tesis doctoral, el interés se centra en el uso de técnicas de metamodelado que especifiquen los conceptos implicados en el proceso de medición. Como se mencionó en la Sección 2.2, en esta tesis se propone un metamodelo que define los conceptos implicados en el proceso de elicitación de requisitos. Uno de los esfuerzos más significativos con respecto al uso de metamodelos en el proceso de medición es presentado en García *et al.* [2] y denominado *Software Measurement Metamodel* (SMM), donde se define una ontología junto con un metamodelo que recoge los conceptos implicados en el proceso de medición tales como medidas, escalas, tipos de escala, etc.

Después de ofrecer a los modeladores el metamodelo de requisitos y dada la existencia del mencionado metamodelo de medición, observamos que el camino natural para ofrecer capacidad para el modelado de requisitos medibles consistía en la integración de ambos metamodelos. De esta forma, los requisitos elicitados quedarían ligados con las medidas que ayudarían a verificar su cumplimiento a la vez que las medidas podrían ser formalmente definidas y quedarían unidas tanto a los requisitos como a los objetivos de más alto nivel que ayudan a satisfacer.

En el Capítulo 4 se mostrará la forrma en la que se ha llevado a cabo esta integración, así como la notación propuesta para el metamodelado de requisitos medibles y la herramienta de soporte creada para este fin. Además, el metamodelo obtenido ha sido integrado en el flujo de desarrollo genérico propuesto por las metodologías de desarrollo de SIW, contribuyendo así a que éstas puedan elicitar requisitos medibles. Concretamente, para realizar esta integración se ha seleccionado la iniciativa MDWEnet [55], un esfuerzo común de la comunidad de Ingeniería Web orientado a unificar los conceptos de la disciplina y facilitar la interoperabilidad entre las diferentes propuestas existentes [56].

2.3. Desarrollo de Software Dirigido por Modelos

El desarrollo dirigido por modelos (*Model Driven Development*, MDD) es un enfoque para la construcción de software en el que los modelos juegan un papel central. En estas propuestas, el desarrollo es conducido por modelos con diferentes niveles de abstracción y transformaciones entre ellos. Aunque los modelos han sido utilizados tradicionalmente en el desarrollo de software, en este paradigma se promueve un cambio en la concepción de los mismos ya que éstos aumentan su protagonismo y pasan de ser simples artefactos para la comunicación y documentación del sistema a verdaderos artefactos software que conducirán la implementación del sistema final.

Uno de los enfoques más extendidos surgido en el ámbito de MDD es la denominada *Model Driven Architecture* (MDA) [57], una instancia concreta de MDD que, promovida por el *Object Management Group* (OMG [58]) desde el año 2001, proporciona una serie de conceptos y guías para la aplicación de un paradigma de desarrollo dirigido por modelos. También en este ámbito, y como una generalización de MDA, surgiría más tarde el concepto de *Model Driven Engineering* (MDE) [5, 59].

Concretamente, la propuesta MDA propone una separación clara entre los aspectos relativos al dominio y reglas de negocio del software a desarrollar frente a los aspectos técnicos y detalles de implementación de éste [57]. Para ello, en MDA se propone llevar a cabo el proceso de desarrollo utilizando modelos con diferentes niveles de abstracción. En el nivel de abstracción más alto se encuentran los Modelos Independientes de la Computación (CIM, *Computation Independent Model*) cuya misión es representar el dominio del sistema a desarrollar y ofrecer una visión de los requisitos del mismo.

En un nivel de abstracción más bajo se encuentran los Modelos Independientes de la Plataforma (PIM, *Platform Independent Model*), en los cuáles se modela la funcionalidad y estructura del sistema pero abstrayéndose todavía de los detalles técnicos de la

plataforma donde éste se implementará. A continuación, la propuesta MDA propone la construcción de Modelos Específicos de la Plataforma (PSM, *Platform Specific Model*), que añaden a los modelos PIM los detalles específicos de la plataforma elegida para implementar el sistema. A partir de los modelos PSM se debe obtener (incluso de forma automática o, al menos, semiautomática siempre que sea posible) la implementación del sistema final.

Como se puede deducir de la explicación anterior, uno de los aspectos clave en el paradigma MDA es el de las transformaciones entre modelos. De forma general, esta transformaciones pueden dividirse en dos grandes grupos [60]:

- Transformaciones verticales: son aquellas que convierten un modelo origen que tiene un determinado nivel de abstracción en un modelo destino con un nivel de abstracción diferente. Por ejemplo, las transformaciones de un PIM a un PSM o de un PSM a código de una plataforma específica serían transformaciones verticales.

- Transformaciones horizontales: son aquellas que convierten un modelo origen en un modelo destino con el mismo nivel de abstracción. Las transformaciones PIM-PIM o PSM-PSM son ejemplos de transformaciones horizontales que pueden tener como objetivo, por ejemplo, añadir nuevos requisitos o restricciones sobre un modelo o solucionar problemas detectados en él.

En los últimos años se ha producido una investigación intensiva en el ámbito del desarrollo dirigido por modelos. Este hecho ha motivado la aparición de múltiples tecnologías, lenguajes y herramientas destinadas a dar soporte a esta propuesta o a alguno de sus componentes. Para la definición de nuevos lenguajes de modelado, el OMG propuso el lenguaje de metamodelado MOF (*Meta Object Facility* [61]). Para dar soporte a la especificación de transformaciones, el OMG definió el estándar QVT (Query/View/Transformations) [62] y también han surgido diferentes entornos y herramientas con este fin como ATL (*Atlas Transformation Language* [63]), RubyTL [64, 65], SmartQVT [66] o MediniQVT [67]. Otras herramientas en este ámbito son IBM Rational Software Design [68], AndroMDA [69], el framework para gestión de modelos MOMENT [70, 71], propuestas como las *Software Factories* de Microsoft [72] o entornos como Eclipse [73].

Las propuestas de desarrollo dirigido por modelos han sido aplicadas con éxito en diferentes ámbitos y organizaciones de reconocido prestigio [11, 74]. Los principales beneficios que se encuentran gracias a su aplicación suelen ser un aumento de la productividad unido a una reducción en los tiempos y costes de desarrollo, una mayor portabilidad de las soluciones ofrecidas y una mejor calidad de los productos desarrollados [11, 36, 75].

Sin embargo, la relativa juventud de la propuesta hace que ésta todavía deba enfrentarse a numerosos retos, existiendo diferentes líneas de investigación abiertas como la definición de lenguajes de modelado completamente ejecutables, el análisis de los modelos utilizados (verificación, validación, análisis de consistencia, etc.) así como también la definición del soporte automático adecuado para los diferentes componentes de la propuesta [11, 76].

En esta tesis se hará uso de las técnicas propuestas por MDD en lo que se refiere al metamodelado y uso de transformaciones y se propondrán mejoras encaminadas a contribuir tanto en el tratamiento dado por este tipo de metodologías a los requisitos como en las líneas de investigación abiertas orientadas al análisis de los modelos utilizados y al uso de MDD en el ámbito de la seguridad, tal y como se detalla en la siguiente sección.

2.3.1. Seguridad dirigida por modelos

La seguridad se ha convertido en un aspecto crucial en el desarrollo de sistemas software. Sin embargo, las estadísticas muestran que cada año se incrementa el número de vulnerabilidades encontradas en estos sistemas, lo que provoca importantes daños para las organizaciones en las que se producen [77]. Una de las principales causas de estas vulnerabilidades es la falta de una adecuada consideración de los requisitos de seguridad del sistema [77]. En multitud de ocasiones, los aspectos de seguridad únicamente son considerados cuando el sistema está completamente desarrollado. Sin embargo, los requisitos de seguridad deben ser adecuadamente tratados en todas las fases del desarrollo del sistema, desde la elicitación de requisitos hasta la implementación del mismo [77, 78, 79].

En los últimos años han aparecido múltiples propuestas centradas en la gestión de los requisitos de seguridad. Por ejemplo, algunas propuestas *goal oriented* para la gestión de requisitos han sido adaptadas con este propósito. Este es el caso de [80], que amplía el framework KAOS [42] para modelar las posibles amenazas sobre el sistema o la metodología Tropos [81], que es extendida con capacidades para el modelado y análisis de requisitos de seguridad en Secure Tropos [82].

Otras técnicas propuestas para la consideración de requisitos de seguridad han sido el uso de ontologías [83], plantillas [84] o propuestas como la de Haley *et al.* [77], que proporciona una serie de actividades para la consideración de requisitos que deja abiertas las técnicas a ser usadas en cada una de ellas.

Las propuestas de desarrollo dirigido por modelos también han influido en la consideración de requisitos de seguridad. En Basin *et al.* [85] se mostró como estas propuestas podían ser especializadas para tratar con las necesidades de seguridad de un sistema y se acuñó el término *Model Driven Security* (MDS) para denominar a este área de trabajo. Concretamente, en Basin *et al.* [85] se mostró la viabilidad del enfoque MDS utilizando modelos para crear artefactos software relacionados con políticas de control de acceso. La propuesta se basaba en el uso de un perfil UML [86] denominado SecureUML.

Otra de las propuestas más relevantes en el ámbito de la seguridad es UMLSec [87], una extensión de UML que permite plasmar aspectos de seguridad sobre los modelos UML utilizados en el desarrollo del sistema y su posterior evaluación utilizando una semántica formal. Recientemente, la propuesta UMLSec también ha evolucionado hacia el paradigma MDS [88].

Tanto UMLSec como SecureUML utilizan perfiles como mecanismo de soporte. Sin embargo, en los últimos años el uso de lenguajes específicos de dominio (*Domain Specific*

Languages, DSL) es una práctica más habitual debido a que ofrecen mejores capacidades de extensión que los perfiles [89, 90]. El metamodelado es un aspecto clave en la construcción de estos DSL, los cuáles están formados por tres elementos principales: un metamodelo para la sintaxis abstracta, una sintaxis concreta y una semántica. La sintaxis abstracta define los conceptos que toman parte en el DSL, las relaciones entre ellos y las restricciones que indican la forma en la que los modelos deben ser creados. La sintaxis concreta define una notación para la sintaxis abstracta y la semántica es generalmente proporcionada a través de transformaciones que transforman un modelo expresado en el DSL a modelos expresados en lenguajes con una semántica bien definida [89]. Siguiendo esta tendencia del uso de DSL, han surgido nuevas propuestas en el ámbito de la seguridad dirigida por modelos como el framework de seguridad OpenPMF [91].

Como comentamos anteriormente, una de las contribuciones de esta tesis doctoral es un metamodelo de requisitos que aglutina los conceptos que habitualmente se utilizan en el proceso de elicitación de requisitos. Este metamodelo ha servido como punto de partida para el desarrollo de una propuesta en el ámbito de MDS, basada en las técnicas y herramientas que actualmente propone el desarrollo dirigido por modelos. Esta propuesta, a la que se ha denominado ModelSec [30], ofrece una arquitectura que permite la generación automática de artefactos software relacionados con la seguridad del sistema (por ejemplo, reglas para implementar políticas de seguridad) partiendo de modelos de requisitos. En la base de esta arquitectura se encuentra el metamodelo de requisitos propuesto que, además, ha sido extendido con conceptos específicos del ámbito de la seguridad como activos, amenazas, contramedidas, etc. En lugar de usar perfiles UML como las propuestas comentadas, se ha construido un DSL gráfico que permite expresar modelos de requisitos de seguridad conformes al metamodelo definido. Además, estos modelos son utilizados como entrada para una cadena de transformaciones que permite la generación de código. Una descripción detallada de estos metamodelos será ofrecida en el Capítulo 4.

2.4. Ingeniería Web

El auge de Internet en la última década ha provocado un incremento exponencial en el número de sitios web. Según las estadísticas proporcionadas por la empresa Netcraf [1], dedicada entre otros cometidos a la investigación sobre el uso de Internet, en el año 2009 existen en la red de redes más de 230 millones de sitios web de los que más de 70 millones se consideran realmente activos. Además, se estima un ritmo de crecimiento que supera los 5 millones de sitios web nuevos cada mes.

En sus inicios, los sitios web eran usados por las organizaciones simplemente para tener presencia en Internet pero la realidad es que estos sistemas son cada vez más complejos y se han convertido en elementos fundamentales para las estrategias de negocio de muchas organizaciones [13]. Para la construcción exitosa de estos sistemas se vio necesario dotar

[1] Netcraf, http://news.netcraft.com/

a los desarrolladores con metodologías adecuadas, procesos disciplinados y repetibles, buenas guías y herramientas de soporte [12]. Para cubrir estas necesidades apareció la disciplina de la Ingeniería Web (*Web Engineering*, WE), la cuál promueve el uso de principios científicos e ingenieriles y aproximaciones sistemáticas para el desarrollo, despliegue y mantenimiento de sistemas web de alta calidad [12].

En el ámbito de la Ingeniería Web han aparecido diferentes metodologías. En sus inicios, y previamente incluso al nacimiento de la Ingeniería Web como disciplina propia, estas metodologías surgieron como adaptaciones de métodos de desarrollo tradicionales que se especializaban o particularizaban a las necesidades concretas de las aplicaciones web. Así surgieron enfoques alineados con el paradigma de desarrollo estructurado como *Hypermedia Design Model* (HDM) [92] o *Relation Management Method* (RMM) [93] que más tarde se extendería con ERMM [94]. Posteriormente, la tendencia varió hacia el paradigma orientado a objetos con propuestas como *Object Oriented Hypermedia Design Method* (OOHDM) [95, 96]. Otros enfoques que surgieron en este ámbito de forma casi contemporánea fueron W2000 [97], UWE [98] o Ariadne [99] así como otras propuestas que tuvieron su origen en el desarrollo de bases de datos como Araneus [100, 101] o WebML [102]. Otras propuestas metodológicas para el desarrollo de sistemas web son WSDM [103], Autoweb [104], el método basado en UML de Conallen [105], OO-H [106, 107] o MIDAS [108]. En la Figura 2.1 (extendida de [1]) se puede ver de forma gráfica la evolución de las metodologías de desarrollo surgidas en este ámbito y las relaciones que hay entre ellas. Como se observa en la Figura 2.1, en los primeros años de la disciplina había una mayor tendencia a la creación de nuevas propuestas mientras que con el paso de los años en lugar de crear nuevos enfoques se tiende a hacer evolucionar los existentes para abordar los nuevos retos surgidos (por ejemplo, nuevos tipos de aplicaciones), adoptar los beneficios aportados por otros paradigmas como el de desarrollo dirigido por modelos y trabajar en esfuerzos comunes que permitan la interoperabilidad entre diferentes propuestas [56].

Recientemente, las propuestas de desarrollo dirigido por modelos también han influido en estas metodologías de desarrollo apareciendo la denominada Ingeniería Web dirigida por modelos (*Model Driven Web Engineering*, MDWE [18]), que ha demostrado aportar beneficios al área de Ingeniería Web. Diferentes propuestas como, por ejemplo, WebML, MIDAS, UWE y OO-H y sus herramientas de soporte, como WebRatio [109, 110], ArgoUWE [111] o M2DAT [112], se han alineado con esta propuesta y proponen un conjunto de modelos y transformaciones entre ellos para llevar a cabo el desarrollo del sistema. Aunque cada metodología presenta características específicas, de forma general, todas ellas suelen utilizar diferentes modelos para tratar con las diferentes vistas del sistema como modelos navegacionales (para modelar la interacción entre los usuarios y el sistema), modelos de comportamiento (para modelar los datos y funcionalidades ofrecidas) y modelos de presentación (en los que se representan características relacionadas con la presentación final del sistema). Para una comparativa detallada acerca de cada una de las metodologías así como de las similitudes y diferencias entre ellas se puede consultar [106, 112, 113].

Con poco más de una década de historia, la Ingeniería Web es todavía una disciplina con múltiples líneas de investigación abiertas que se debe enfrentar, tanto actualmente como

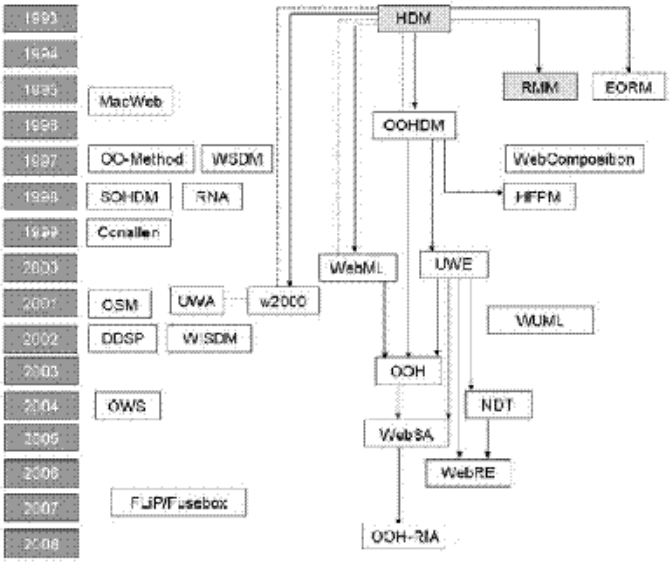

Figura 2.1: Metodologías surgidas en Ingeniería Web (extendido de [1])

en el futuro, a numerosos retos debido, por ejemplo, al crecimiento tanto en número como en complejidad de los sistemas web o la necesidad de construir sitios web universalmente accesibles, con alta disponibilidad, escalabilidad, usabilidad, fiabilidad y preparados para diferentes tipos de dispositivos como ordenadores o dispositivos móviles [14].

Varias son las líneas de trabajo de esta tesis que pueden ser aplicadas en este ámbito. El tratamiento adecuado de los requisitos en este tipo de proyectos de desarrollo es una de estas líneas [1]. Otra línea de trabajo está relacionada con la usabilidad de las aplicaciones web, ya que se hace necesaria la definición de métodos para asegurar que las aplicaciones desarrolladas tengan la usabilidad deseada [114] así como mecanismos para la evaluación de este atributo en fases iniciales del desarrollo [19]. La misma necesidad existente con respecto a la usabilidad surge en relación con la seguridad de este tipo de software [21, 38]. Otra línea de trabajo más que, de nuevo, no es exclusiva de la Ingeniería Web sino general de todos los enfoques alineados con el desarrollo dirigido por modelos, está relacionada con la introducción de mecanismos para el análisis de los modelos usados como técnicas de verificación, validación o análisis de consistencia [36], aunque en el marco de esta tesis serán ilustradas usando este dominio.

En las siguientes subsecciones mostraremos el estado de la investigación en estas líneas de trabajo y motivaremos las aportaciones que sobre ellas se hacen en esta tesis.

2.4.1. Ingeniería de Requisitos en Ingeniería Web

De la misma forma que en cualquier otro tipo de proyecto software, la gestión adecuada de los requisitos es clave para el éxito de los proyectos de Ingeniería Web. Diferentes estudios centrados en este ámbito resaltan la importancia que tienen las actividades

de IR en el éxito de este tipo de proyectos así como la necesidad de un proceso de IR incluso más detallado que el realizado para otros tipos de aplicaciones [13, 17, 37]. Sin embargo, la introducción de técnicas para el adecuado manejo de los requisitos es una línea de investigación abierta en la disciplina ya que la mayoría de las metodologías existentes se encuentran centradas principalmente en aspectos relacionados con el diseño del sistema, dejando de lado el tratamiento de los requisitos a pesar de los riesgos que ésto conlleva [1, 37, 115]. Algunas de las conclusiones obtenidas por estos estudios fueron las siguientes:

- Muchas de las propuestas de desarrollo no incluyen ningún tipo de etapa o recomendación para el tratamiento de los requisitos. Esto ocurría sobre todo en las primeras metodologías que aparecieron como RMM o HDM, pero también en otras más recientes como WSDM o el método Conallen u otras como OOHDM que no consideraban estos aspectos en sus primeras versiones pero que sí los añadieron posteriormente.
- En cuanto a las metodologías que proponen actividades relacionadas con IR, la técnica mayoritaria para la captura de requisitos consiste en la utilización de casos de uso de UML. Éste es el caso de metodologías como OO-HDM, UWE, WebML o W2000. Sin embargo, estudios como el realizado por Insfrán *et al.* [116] han demostrado que los casos de uso son inadecuados y demasiado ambiguos para este fin y, concretamente, para especificar detalles como las características de navegación del sistema. Es por ello que metodologías como OOHDM o UWE han introducido otras características para complementar a estos casos de uso como plantillas, diagramas de actividades de UML o *User Interaction Diagrams* (UIDs [117]).
- En los últimos años, la MDWE ha llevado a las metodologías de desarrollo a potenciar el uso de modelos, metamodelos y transformaciones y ésto ha influido también en la forma de tratar con los requisitos. Siguiendo esta tendencia han aparecido algunas aproximaciones basadas en el uso de metamodelos de requisitos como [118, 119, 120] o NDT (*Navigational Development Techniques* [1], que actualiza la propuesta realizada en [120]).

Estas últimas propuestas para el metamodelado de requisitos son las más relacionadas con el trabajo presentado en esta tesis. Si analizamos los metamodelos de requisitos presentados por *Escalona et al.* [1, 120] se observa que éstos siguen un enfoque orientado al diseño tal y como suele ocurrir en la mayoría de propuestas de Ingeniería Web. Este hecho hace que los conceptos que aparecen en el metamodelo estén demasiado próximos al diseño del sistema (por ejemplo, se incluyen características de visualización o actividades de búsqueda como elementos que participan en las actividades de elicitación de requisitos). El principal problema que ésto conlleva es la ausencia de un grado de expresividad mayor a la hora de definir requisitos, ya que los requisitos de diseño son difíciles de entender y expresar para aquellos *stakeholders* que no están directamente relacionados con el diseño del sistema. Los *stakeholders* generalmente necesitan una forma más abstracta de expresar sus requisitos, es decir, una forma más próxima al dominio bajo en el que la aplicación está siendo desarrollada.

	Tipo de Requisitos	**Considera *goals***	**Reuso de Requisitos**	**Herramienta de soporte**
Escalona y Aragón [1]	Próximos al diseño	No	No	Sí
Bolchini y Paolini [118]	Próximos al diseño	Sí	No	No
Valderas y Pelechano [119]	Próximos al diseño	No	No	Sí
Molina y Toval [22]	Carácter general	Sí	Sí	Sí

Tabla 2.1: Comparativa entre propuestas de metamodelado de requisitos en Ingeniería Web

Esta necesidad de capturar metas y requisitos de usuario con un mayor nivel de abstracción es resaltada por Bolchini y Paolini [118], donde se presenta un enfoque orientado a metas para modelar requisitos de sistemas web. El concepto de meta (*goal*) aparece en el ámbito del framework *i** [121] con el propósito de modelar un objetivo de alto nivel de uno o más *stakeholders*. Los conceptos del framework *i** resultan muy útiles para modelar objetivos de los usuarios aunque su generalidad sugiere la necesidad de adaptarlos a las necesidades de dominios específicos. Siguiendo esta tendencia, [118] utiliza *i** como base pero lo especializa con el propósito de diseñar un nuevo metamodelo de requisitos que recoge conceptos particulares de Ingeniería Web como, por ejemplo, una taxonomía de requisitos web.

De la misma forma que en la propuestas de Escalona *et al.* [1, 120], la taxonomía de requisitos propuesta por [118] está de nuevo muy cercana a la forma en la que se llevaba a cabo el diseño de la aplicación en el momento de la propuesta. Además, los autores de [118] resaltan la necesidad de mejorar el soporte automático para la propuesta así como la necesidad de tratar con el concepto de reutilización de requisitos. Algo similar ocurre con la propuesta de Valderas y Pelechano [119] en la que se propone un metamodelo basado en tareas que de nuevo está demasiado próximo al diseño del sistema, ya que incluye aspectos relacionados con la interacción entre el usuario y el sistema así como con la navegación en el mismo.

En la Tabla 2.1 se muestra una comparativa entre estas propuestas de metamodelado y la presentada en esta tesis, cuyas características aparecen en la última fila de la tabla. Los conceptos considerados en este nuevo metamodelo no estarán tan próximos a los aspectos de diseño del sistema ya que, generalmente, éstos son ajenos a los *stakeholders*, sino que estarán más cercanos al dominio de éstos, facilitándoles así la tarea de expresar sus requisitos y metas. Este metamodelo y su herramienta de soporte ofrecerán la posibilidad de considerar aspectos como la elicitación de objetivos de alto nivel a través del concepto de *goal* y la reutilización de requisitos [122]. Estas contribuciones serán analizadas en detalle en el Capítulo 4.

2.4.2. Evaluación de usabilidad en fases tempranas de desarrollo

De la misma forma que en cualquier otro sistema software, los sistemas basados en web deben tener en cuenta diferentes atributos de calidad como la fiabilidad, robustez, rendimiento, etc. que son claves para su éxito. Además, los sistemas basados en web presentan determinadas particularidades con respecto a otros tipos de software. Murugesan [14] presenta una lista detallada de estas características diferenciadoras entre las que podemos mencionar, por ejemplo, que las aplicaciones web suelen ser usadas por una amplia variedad de usuarios con diferentes requisitos, expectativas y habilidades, que estos sistemas pueden presentar una gran variedad de contenido (texto, imágenes, audio y vídeo) y pueden ser visualizados en diferentes dispositivos con características y velocidades de acceso diferentes, que se precisa de una interfaz con buenos principios estéticos y a través de la que sea fácil navegar o que los tiempos de desarrollo asignados a estos proyectos suelen ser más cortos que para proyectos software tradicionales.

En este ámbito, los usuarios esperan aplicaciones que sean cada vez más fiables, seguras, personalizables, sensibles al contexto y usables, lo que implica que ciertos atributos de calidad adquieran mayor importancia [14, 19]. Uno de estos atributos de calidad que cobra especial relevancia es la usabilidad [19]. Existen diferentes definiciones de usabilidad en la literatura. Por ejemplo, el estándar ISO 9126-1 sobre calidad de productos software [54], define la usabilidad como "la capacidad de un producto software de ser entendido, aprendido, usado y atractivo para el usuario, cuando éste es usado bajo condiciones específicas". Nielsen [123] la define como "una medida de la capacidad de un sistema software para ser aprendido y recordado, de su eficiencia de uso, así como de su habilidad para evitar errores y gestionar tanto errores como la satisfacción de sus usuarios". El estándar ISO 9241-11 en su sección relativa a usabilidad [124] la define como "la medida en la cuál un producto puede ser usado por unos usuarios especificados para conseguir unas metas especificadas con efectividad, eficiencia y satisfacción en un determinado contexto de uso".

La consideración de la usabilidad del software reporta beneficios como la mejora en la productividad de los usuarios o la reducción en los costes de aprendizaje, entrenamiento y documentación [123]. Además, las inversiones realizadas en usabilidad han demostrado devolver importantes beneficios económicos [125, 126], lo cuál ha motivado que importantes organizaciones como IBM o Boeing Co. consideren la usabilidad como un factor relevante en sus productos software. Otro ejemplo muy llamativo sobre estos beneficios es documentado en [127], donde se muestra un caso de estudio en el que los cambios en el diseño de una aplicación web para hacerla más usable de cara a los usuarios reportaron un incremento en los beneficios de ventas que ascendió a los 300 millones de dólares en un sólo año. Este hecho da buena muestra de cómo la usabilidad puede ser crítica de cara al éxito de un sistema software de este tipo.

Para contribuir en la medida de lo posible a la consideración de usabilidad han surgido estándares [54], guías de accesibilidad web [128] o recomendaciones de expertos [123] que reúnen propiedades que los sistemas deben cumplir para mejorar esta característica. También se han hecho esfuerzos orientados a obtener métricas que sirvan para evaluar

la calidad de los SIW desarrollados [129, 130], estando un alto porcentaje de ellas relacionadas con la usabilidad del sistema.

Sin embargo, la consideración de usabilidad en el desarrollo de software se enfrenta con varios obstáculos. Uno de ellos está relacionado con la falta de atención prestada a los requisitos (incluyendo requisitos de usabilidad), en fases tempranas del ciclo de desarrollo. Seffah [131] destaca otros problemas como que las actividades relacionadas con usabilidad se encuentran generalmente desacopladas del proceso de desarrollo de software, el uso de notaciones y herramientas para considerar usabilidad diferentes a las usadas en el proceso establecido para el desarrollo o la falta de soporte automático para algunas actividades relacionadas con usabilidad.

Un análisis de las metodologías y herramientas tradicionalmente usadas para el desarrollo de SIW revela que estos obstáculos continúan estando presentes en el ámbito de la Ingeniería Web. Por ejemplo, las metodologías existentes no suelen preocuparse de atributos de calidad como la usabilidad durante el proceso de desarrollo. Esta tarea es generalmente retrasada hasta que el sistema ha sido completamente desarrollado, usando para ello métodos como el análisis con usuarios o herramientas denominadas *validadores de usabilidad y accesibilidad* que validan el código HTML y CSS del sistema (ver [132] para obtener información detallada de estos métodos como por ejemplo [133, 134, 135]). Por tanto, la posibilidad de desplazar algunas de estas validaciones hacia fases más tempranas del ciclo de desarrollo como el modelado y su integración en las metodologías de desarrollo no están siendo consideradas.

Otro ejemplo de los problemas mencionados por Seffah [131] puede observarse en el uso de métricas de calidad dentro del ámbito de desarrollo de SIW. En Ruiz *et. al* [129] se llevó a cabo una investigación sobre estas métricas, identificándose hasta 385 métricas que habían sido definidas con el propósito de medir atributos de calidad sobre este tipo de sistemas. La mitad de estas métricas estaban relacionadas con usabilidad pero, como argumenta [129], éstas generalmente no están definidas de una forma precisa y no son soportadas por ninguna herramienta o, si este soporte existe, es ofrecido por herramientas externas que son independientes de aquellas usadas en el proceso de desarrollo de SIW.

Como causa de esta problemática, tal y como se indica en Matera *et. al* [114], la definición de métodos que ayuden a asegurar la usabilidad de las aplicaciones basadas en web es uno de los objetivos de investigación que continúan abiertos tanto en Ingeniería Web como en la disciplina de Interacción Persona-Ordenador (*Human Computer Interaction*, HCI). Además, estudios como los de Juristo *et al.* [19, 20] indican que la consideración de atributos de usabilidad debe ser desplazada, siempre que sea posible, hacia fases iniciales del ciclo de vida del proyecto, lo que permitirá conseguir beneficios como la mejora en la satisfacción de los usuarios, la detección temprana de problemas y la reducción del tiempo y dinero invertido tanto en la solución de esos problemas como en el posterior mantenimiento de la aplicación [19, 136, 137].

En esta tesis doctoral se trabajará en el área de evaluación de usabilidad introduciendo mejoras que permitan considerar la definición de requisitos relacionados con usabilidad desde fases tempranas del desarrollo. Con este objetivo se propone, en primer lugar, hacer un mayor hincapié en la elicitación de requisitos de usabilidad utilizando para ello los

metamodelos de requisitos presentados en el Capítulo 4. A continuación, en el Capítulo 5, se mostrará una propuesta de extensión de los modelos usados en el desarrollo de sistemas web con el objetivo de que sea posible reflejar sobre ellos requisitos de usabilidad y se introducirán técnicas que permitan la evaluación de requisitos y métricas de calidad durante el modelado del sistema.

2.4.3. Análisis de los modelos utilizados en el desarrollo de sistemas web

Como se mencionó en secciones previas, en las propuestas *model driven* los modelos del sistema pasan a ser un elemento clave en el proceso de desarrollo y la calidad de éstos se convierte en crítica de cara al éxito del sistema implementado [6]. Sin embargo, la introducción de técnicas para el análisis de estos modelos es todavía una línea abierta que necesita ser reforzada en todas las metodologías *model driven*, incluyendo también aquellas específicas del ámbito de la Ingeniería Web [11, 76].

En Ingeniería del Software se han propuesto multitud de técnicas que tienen por objeto analizar y contribuir a mejorar la calidad de los modelos conceptuales del software como son las técnicas de verificación y validación (V&V) de modelos o las de *model checking* [138]. Sin embargo, tal y como se resalta en Alpuente *et al.* [139], han sido pocos los trabajos orientados a la verificación de sitios web. Además, un análisis de las metodologías de desarrollo de SIW revela que éstas no suelen incorporar mecanismos para verificar y comprobar la calidad de los modelos utilizados para el desarrollo del sistema [32].

Uno de los enfoques que surgieron para solucionar esta carencia consiste en la aplicación de técnicas formales en el ámbito de la Ingeniería Web y, concretamente, el uso de técnicas como la reescritura de términos [140] y la especificación algebraica de SIW. Este tipo de técnicas comenzaron a aplicarse en trabajos como [139, 141, 142, 143] y también como parte de esta tesis doctoral [31, 32], realizándose una propuesta para la especificación algebraica de SIW y verificación de propiedades sobre ellos, que será explicada de forma detallada en el Capítulo 6. Desde la aparición de estos primeros trabajos, las técnicas formales han venido siendo utilizadas en el ámbito de la Ingeniería Web en tareas como V&V [144, 145], *model checking* [146] o consideración de aspectos de seguridad [147].

La aplicación de este tipo de técnicas ha contribuido a la mejora de los sistemas desarrollados. Sin embargo, existen determinados aspectos en los que es necesario seguir trabajando para conseguir la integración de las técnicas formales en las metodologías y procesos de desarrollo. Por un lado, algunas propuestas como [142, 143, 145] realizan tareas de verificación sobre el código del sistema en lugar de sobre artefactos propios de fases más tempranas del desarrollo como los modelos conceptuales. Por otro lado, estos enfoques formales generalmente no disponen de un soporte automático o éste es proporcionado por herramientas aisladas que no se encuentran integradas en los procesos de desarrollo de sistemas web. Además, el uso de técnicas formales suele requerir un conocimiento profundo acerca de la sintaxis y semántica de sus conceptos, los cuáles

son completamente ajenos a los modeladores web que están acostumbrados a pensar en términos de páginas, enlaces, etc. [146].

Estos problemas dificultan en gran medida la aplicabilidad de técnicas formales en el ámbito de la Ingeniería Web, impidiendo por tanto su adopción en entornos industriales así como que los modeladores obtengan las ventajas de su aplicación. Por ello, como se indica en [148], la integración de estas técnicas es uno de los mayores retos y tendencias en el uso de métodos formales, es decir, integrar este tipo de técnicas en entornos uniformes que, a su vez, puedan ser integrados en plataformas estándar de desarrollo como Eclipse [73]. Con este objetivo en mente se está trabajando en enfoques y herramientas de soporte como las desarrolladas en el ámbito del proyecto Sensoria [149, 150].

En el Capítulo 6 se propone una aproximación para la especificación algebraica de sistemas web y se explica la forma en la que se ha contribuido en la medida de lo posible a la integración del uso de técnicas formales en Ingeniería Web a través de un entorno que permite la construcción de modelos, su transformación a especificaciones algebraicas y la aplicación de técnicas formales sobre ellos con propósitos de verificación a través de un soporte automático implementado en Eclipse que oculta a los modeladores todos los detalles del formalismo subyacente.

Otro de los problemas que suele surgir durante el desarrollo dirigido por modelos es el de la consistencia entre éstos. Los problemas de consistencia han existido desde siempre en el desarrollo de sistemas de información, estando ligados generalmente a la existencia de diferentes modelos o vistas de un mismo sistema que pueden ser el origen de numerosos errores en el software generado [151]. En una revisión sistemática que hemos realizado con el objetivo de analizar las propuestas existentes para análisis de consistencia se concluyó que en este área se habían realizado muchos trabajos de gran calidad pero que, al mismo tiempo, todavía existían líneas abiertas que debían ser tratadas [8]. Las principales carencias detectadas en [8] estaban relacionadas con la falta de mecanismos de extensión en las propuestas analizadas (con lo que resultaba difícil añadir y gestionar nuevos problemas de consistencia cuando éstos eran detectados), la necesidad de mejorar el soporte automático para las propuestas existentes y la necesidad de integrar mecanismos de análisis de consistencia en propuestas de desarrollo dirigido por modelos, donde especialmente el análisis de consistencia entre modelos con diferentes niveles de abstracción debía ser reforzado. Las necesidades en el ámbito del desarrollo dirigido por modelos resaltadas en [8] están en consonancia con las carencias indicadas en [11, 76], donde también se pone de manifiesto la importancia de introducir técnicas para el análisis de los modelos utilizados.

Los mencionados problemas de consistencia se dan también en el desarrollo de SIW ya que, como se mencionó anteriormente, las metodologías de desarrollo proponen diferentes modelos del mismo sistema como modelos navegacionales, de comportamiento o de presentación. Además, en este ámbito también han surgido modelos para evaluar atributos de calidad como el presentado en [22], que añaden una nueva vista al sistema que potencialmente podría introducir nuevos problemas de consistencia. Los problemas de análisis de consistencia no han sido específicamente tratados en el ámbito de la Ingeniería Web así que se está abordando una aproximación inicial para el análisis de consisten-

cia entre modelos de este ámbito que ha sido presentada en [33] y que utiliza modelos navegacionales y los modelos para evaluación temprana de usabilidad presentados en [22].

2.5. El lenguaje Maude

Como comentamos en la Sección 2.4, en esta tesis doctoral se hará uso de técnicas formales para dar soporte a actividades de verificación y análisis de modelos. El lenguaje formal utilizado para definir estas actividades es Maude [152], un lenguaje de especificación formal basado en lógica ecuacional y lógica de reescritura con especificaciones ejecutables que permiten la construcción de prototipos, el chequeo de restricciones o la prueba de propiedades de una forma eficiente. Maude es una extensión de OBJ3 que proporciona, entre otras funcionalidades, herencia múltiple, programación parametrizada y programación modular, facilitando así la escalabilidad de la especificación y la manipulación adecuada de la complejidad de un sistema. Maude también permite especificar sistemas basados en objetos distribuidos y concurrentes.

En lógica de reescritura un sistema es especificado a través de una teoría de reescritura, consistiendo ésta en una signatura Σ (tipos y operaciones), un conjunto E de ecuaciones y un conjunto de reglas de reescritura. La parte estática del sistema es modelada por medio de lógica ecuacional (Σ y E) y la parte dinámica es especificada añadiendo reglas de reescritura que se encargan de definir cómo el estado del sistema cambia. Una regla de reescritura (a la que llamaremos l) describe una transición que tiene lugar en el sistema. Si el patrón especificado en la parte izquierda de la regla (t) se corresponde con un fragmento del estado del sistema, ese fragmento es transformado en la parte derecha de la regla (t'), lo cuál es expresado como $l : t \rightarrow t'$.

En cuanto a la estructura, el concepto clave en Maude es el de módulo. Un módulo es, básicamente, un conjunto de definiciones. En un módulo se define una colección de tipos y una colección de operaciones sobre dichos tipos, además de la información necesaria para reducir y reescribir las expresiones que introduzca el usuario. En Maude existen tres tipos de módulos: funcionales, del sistema y orientados a objetos. Los módulos funcionales son teorías en lógica ecuacional de pertenencia. Los módulos del sistema están basados en lógica de reescritura. Los módulos orientados a objetos permiten manejar los conceptos clásicos dentro de este paradigma tales como clases, atributos, mensajes, objetos, etc. Una descripción más detallada acerca de este lenguaje y sus características puede encontrarse en [152].

En el Capítulo 6 se ofrecerá más detalle acerca del uso que se ha hecho de este lenguaje en el contexto de esta tesis doctoral.

2.6. La plataforma Eclipse

La plataforma Eclipse [73] está siendo ampliamente adoptada para la implementación e integración de las herramientas surgidas en el ámbito del desarrollo de software diri-

gido por modelos (por ejemplo, para la construcción de DSL [89] o *plug ins* para otras herramientas). Siguiendo esta tendencia, esta plataforma ha sido seleccionada para la implementación del soporte automático para las contribuciones realizadas en esta tesis doctoral.

La aceptación de Eclipse como plataforma de referencia por la comunidad de desarrollo dirigido por modelos está motivada por diferentes razones. Eclipse es una plataforma de código abierto (*open source*) basada en Java que proporciona una gran cantidad de herramientas de desarrollo. Una de las claves para su éxito radica en que proporciona una arquitectura flexible basada en el uso de *plug ins* que facilita el uso, creación, reutilización y extensión de herramientas software.

Eclipse, a través de sus diferentes proyectos, proporciona implementaciones para las especificaciones propuestas por el OMG. Por ejemplo a través de EMF (*Eclipse Modelling Framework*) se permite la creación y gestión de metamodelos Ecore, incluyéndose además un editor gráfico que permite la construcción de metamodelos de forma visual, como si se tratara de metamodelos MOF [61]. Sobre EMF descansa GMF (*Graphical Modelling Framework* [153]), que permite la creación de editores gráficos para la gestión de los elementos que componen los metamodelos definidos. Además, Eclipse también ofrece mecanismos de acceso sencillo a la representación de sus modelos y metamodelos.

Por otro lado, también se han integrado en Eclipse diferentes propuestas para ofrecer soporte a las transformaciones entre modelos. Éste es el caso de herramientas para el soporte a transformaciones modelo a modelo como las mencionadas en la Sección 2.3 o lenguajes para el soporte de transformaciones modelo a código como MOFScript [154], un lenguaje de plantillas ampliamente extendido.

En los siguientes capítulos se mostrará el uso que se ha hecho de Eclipse para ofrecer soporte a las diferentes contribuciones presentadas.

Capítulo 3

Método para la mejora de los artefactos software creados en fases tempranas del desarrollo

3.1. Introducción

En este capítulo se ofrece una descripción general del método propuesto en esta tesis. Como se mencionó en el Capítulo 1, este método servirá como marco para la integración de un conjunto de contribuciones orientadas a la mejora de los artefactos software obtenidos en fases iniciales de desarrollo del sistema y que serán explicadas en los capítulos posteriores. En primer lugar, la Figura 3.1 muestra una primera visión general en la que se indica la forma en la que estas contribuciones se integran en un ciclo clásico de desarrollo de software, estando éstas enmarcadas en las etapas de gestión de requisitos y modelado conceptual del sistema.

Para reforzar la consideración de los requisitos se ha diseñado un metamodelo de requisitos que ha sido extendido tanto con un metamodelo que permite la elicitación de medidas para evaluar estos requisitos como con un metamodelo que aglutina conceptos relacionados con la seguridad del sistema y que incide en la elicitación temprana de este tipo de requisitos. Posteriormente, los requisitos obtenidos en esta etapa ayudarán a modelar de forma temprana aspectos como la seguridad y usabilidad del sistema.

Además, la etapa de modelado conceptual será complementada con actividades para el análisis de los modelos construidos con el fin de detectar posibles errores (por ejemplo, propiedades incumplidas relacionadas con requisitos de usabilidad o problemas de consistencia entre modelos) o mejoras (por ejemplo, a través del cálculo de métricas de calidad) que contribuyen a la construcción de mejores modelos y evitan que problemas en éstos sean propagados hacia fases posteriores del ciclo de vida en los que su solución sería más costosa.

A continuación, se describirá con más detalle el método propuesto. Para esta descripción

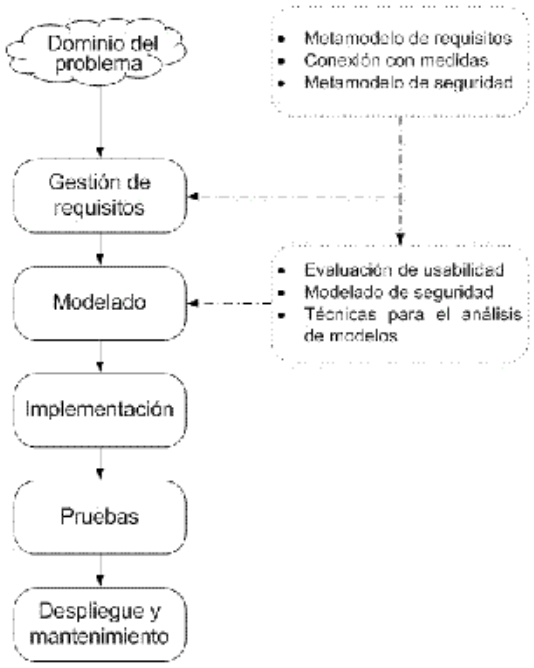

Figura 3.1: Integración de contribuciones en un ciclo clásico de desarrollo de software

se utilizará la notación definida por el estándar SPEM (*Software Process Engineering Metamodel* [155]), promovido por el OMG.

La Figura 3.2 muestra una visión general de este método, que ha sido dividido en cuatro fases compuestas a su vez de varias actividades que se irán detallando en las siguientes secciones. A grandes rasgos, la primera fase pretende reforzar la elicitación de requisitos mediante el uso de técnicas de metamodelado. La segunda fase complementa a la fase de modelado conceptual del sistema y se centra en el modelado de requisitos de usabilidad y seguridad. La tercera fase se enfoca hacia el análisis de los modelos conceptuales previamente creados mediante actividades como la evaluación de requisitos y métricas de usabilidad o la verificación formal de estos modelos. Por último, toda la información obtenida tras este análisis servirá como *feedback* para una fase de mejora que permitirá incrementar la bondad de los modelos de requisitos y conceptuales creados inicialmente antes de que éstos avancen hacia fases posteriores del ciclo de vida. En las siguientes secciones se explicarán tanto estas fases como las actividades que las componen de forma detallada.

3.2. Fase 1: extracción de requisitos y medidas

El objetivo de esta fase consiste en descubrir y representar los requisitos del sistema software que debe ser desarrollado. En el marco de esta tesis se han desarrollado tres contribuciones que se integran en esta fase, siendo usada cada una de ellas en una de

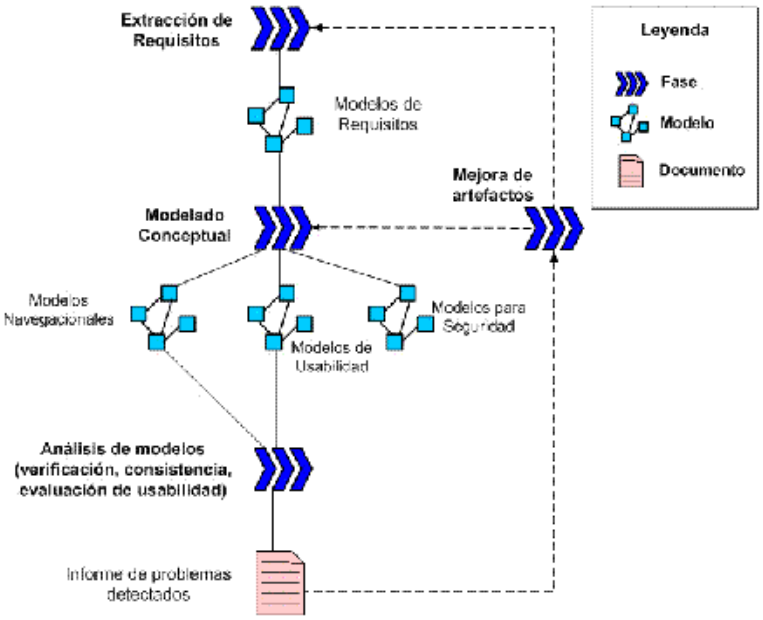

Figura 3.2: Proceso abstracto de mejora de la calidad en fases tempranas de desarrollo

las tres actividades que la componen y que se muestran en la Figura 3.3. Tras la ejecución de esta fase se deberá disponer de modelos de requisitos del sistema que, a su vez, podrán estar asociados con medidas que contribuyan a la validación del cumplimiento de los mismos, así como modelos específicos para detallar los requisitos de seguridad, en caso de que sean necesarios. Todos estos modelos deberán ser conformes a los metamodelos previamente definidos, los cuáles será descritos en profundidad en el Capítulo 4. A continuación se detalla el propósito de cada una de estas actividades.

3.2.1. Actividad 1.1: elicitación *genérica* de requisitos

En esta actividad se hará uso del metamodelo de requisitos diseñado al que se ha denominado URM (*Unified Requirements Metamodel*). Este metamodelo define los conceptos que participan en el proceso de elicitación de requisitos y refuerza la consideración de éstos en metodologías de desarrollo alineadas con el enfoque MDD. Esta actividad recibe el nombre de elicitación *genérica* de requisitos por ser independiente del tipo de requisitos elicitados, a diferencia de la siguiente actividad que se centra en la elicitación de requisitos específicamente relacionados con la seguridad del sistema. El metamodelo definido, que viene acompañado de una notación y una herramienta de soporte, ayudará a los *stakeholders* a identificar aspectos como las metas (*goals*) de su sistema y los requisitos necesarios para alcanzarlas, que podrán ser descritos en base a términos definidos de forma precisa que faciliten su comprensión. Este metamodelo y todos los conceptos que lo componen serán explicados en detalle en el Sección 4.4.

Como resultado del desarrollo de esta actividad se obtendrán modelos de requisitos conformes al metamodelo propuesto en los que se identifiquen y representen las necesidades

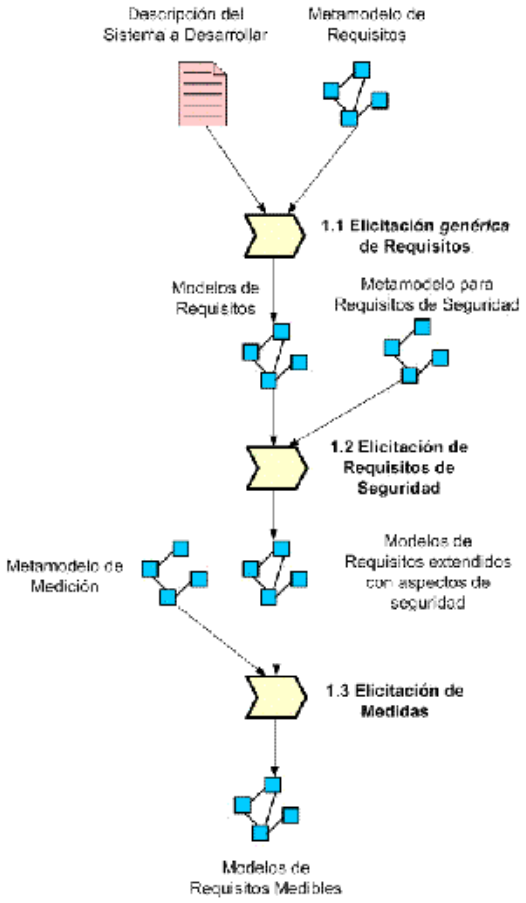

Figura 3.3: Fase 1: Elicitación de requisitos y medidas

de los *stakeholders*. Las técnicas usadas para descubrir estos requisitos se dejan abiertas, es decir, podrían usarse entrevistas, cuestionarios, técnicas de *brainstorming*, etc. (ver por ejemplo [37] para una descripción completa acerca de estas técnicas). Además, en caso de que sea necesario, la descripción de algunos requisitos podría complementarse mediante el uso de otras técnicas como, por ejemplo, especificaciones formales o notaciones específicas.

3.2.2. Actividad 1.2: elicitación de requisitos de seguridad

El metamodelo de requisitos usado en la *Actividad 1.1* ha sido extendido con otro metamodelo diseñado específicamente para la consideración de requisitos de seguridad. Usando esta extensión, aquellos requisitos extraídos en la *Actividad 1.1* y relacionados con la seguridad pueden ser descritos de forma más detallada en base a conceptos propios de este área como los activos que deben ser protegidos, las amenazas contra las que protegerlos o los mecanismos de control de acceso a estos activos. Como resultado de esta actividad se obtendrán modelos de requisitos de seguridad conformes al meta-

modelo propuesto y que será detallado en el Capítulo 4. Si el sistema en desarrollo no tuviera requisitos específicos de este tipo, se pasaría directamente a la *Actividad 1.3* que recibiría como entrada los modelos de requisitos construidos en la *Actividad 1.1*.

3.2.3. Actividad 1.3: elicitación de medidas

Tal y como se explicará en el Capítulo 4, el metamodelo de requisitos comentado en la actividad anterior ha sido integrado con un metamodelo de medición con el objetivo de que, siempre que sea posible, los requisitos sean elicitados junto a medidas que permitan comprobar su cumplimiento, dando así soporte al concepto de requisitos medibles explicado en la Sección 2.2.1. Para aquellos requisitos que sea posible, en esta actividad se deberán elicitar medidas, dando como resultado modelos de requisitos medibles. En el caso de que determinados requisitos hayan sido utilizados en proyectos previos, se podrá hacer uso del concepto de reutilización de requisitos [122], acelerándose así tanto el proceso de extracción de requisitos como de las medidas asociadas a ellos.

3.3. Fase 2: modelado de usabilidad y seguridad

Una vez obtenidos los requisitos, la fase de modelado se centra en la construcción de los modelos conceptuales que permitan la posterior implementación del sistema que los satisfaga. En el marco de esta tesis no se pretende proporcionar una fase de modelado completamente nueva, sino complementar fases tradicionales proponiendo tanto extensiones sobre los modelos usados como nuevos modelos que se integren con éstos y que, en ambos casos, incidan en el modelado de los requisitos de usabilidad y seguridad del sistema. En la Figura 3.4 se muestran las actividades enmarcadas en esta fase, las cuáles son explicadas en las siguientes secciones.

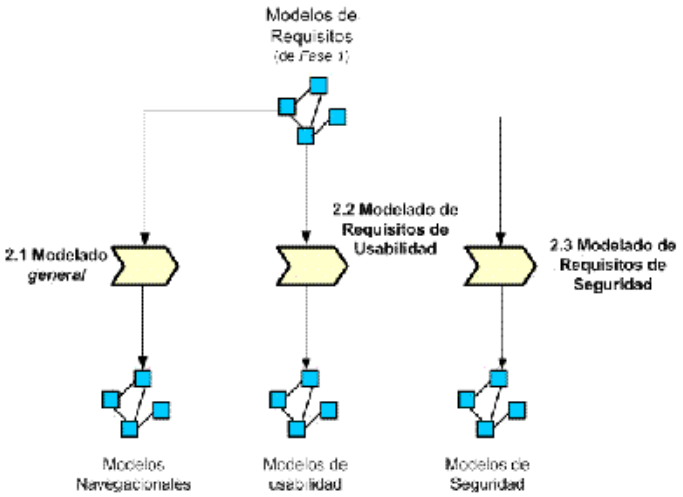

Figura 3.4: Fase 2: modelado específico de requisitos de usabilidad y seguridad

3.3.1. Actividad 2.1: modelado *general* del sistema

Con esta actividad se representa de forma genérica al conjunto de actividades que cada metodología de desarrollo proponga para la obtención de los modelos conceptuales del sistema. Las técnicas desarrolladas en esta tesis complementarán la fase de modelado en dos sentidos. En primer lugar, esta fase puede beneficiarse de las actividades orientadas a reforzar la consideración de requisitos llevadas a cabo durante la *Fase 1*, recibiendo como entrada los modelos de requisitos allí creados. A continuación, la fase de modelado se complementará con contribuciones específicamente orientadas al modelado de requisitos relacionados con la usabilidad y seguridad del sistema y que serán detalladas en la siguientes secciones. La elección de este tipo de requisitos, tal y como se justificó en la Capítulo 2, viene motivada por ser éstos críticos en los sistemas software actuales, especialmente aquellos basados en el uso de la web. Sin embargo, una aproximación similar podría ser llevada a cabo para otros tipos de requisitos considerados importantes en el dominio de la aplicación (por ejemplo, requisitos de interfaz).

Como resultado de esta actividad se obtendrán los modelos conceptuales del sistema, habiéndose destacado de entre estos en la Figura 3.4 los modelos navegacionales que se obtendrían en el caso de una metodología de desarrollo de sistemas web, ya que éstos serán utilizados como parte de la propuesta realizada para la consideración temprana de requisitos de usabilidad descrita a continuación.

3.3.2. Actividad 2.2: modelado de requisitos de usabilidad

Para ilustrar las extensiones a la fase de modelado relacionadas con requisitos de usabilidad se ha utilizado el ámbito de la Ingeniería Web. En el Capítulo 2 se mencionó que aunque existen diferencias entre distintas metodologías, de forma general, todas ellas proponen que en esta fase se obtengan un conjunto de modelos que se centren en los diferentes aspectos del sistema. Básicamente, se utilizan modelos de comportamiento para tratar con los datos y funcionalidades del sistema usando artefactos como modelos de clases UML que, por tanto, son similares a los usados para modelar otros tipos de sistemas pero, además, se añaden otros artefactos más orientados al modelado de aspectos especialmente importantes para los sistemas basados en web. Este es el caso de los modelos navegacionales, centrados en la interacción entre el sistema y sus usuarios o los modelos de presentación, que se centran en los detalles relativos a la forma en la que el sistema será mostrado a sus usuarios.

Debido a que modelan la interacción entre los usuarios y el sistema, los modelos navegacionales construidos influirán en la usabilidad del sistema que finalmente se desarrolle a partir de ellos. Por ello, en esta tesis se propone una extensión de los mismos que permite la expresión de ciertos requisitos de usabilidad obtenidos durante la *Fase 1* sobre ellos y que será detallada en el Capítulo 5. Como resultado de esta actividad se obtendrá un nuevo modelo al que hemos denominado *modelo de usabilidad* en el que se reflejan requisitos de usabilidad que deben ser cumplidos por los modelos navegacionales y que serán evaluados en la *Fase 3*.

3.3.3. Actividad 2.3: modelado de requisitos de seguridad

Para el caso del modelado de la seguridad del sistema, se ha propuesto un nuevo modelo que permite detallar los requisitos de seguridad elicitados en la *Actividad 1.2* a un nivel de abstracción más bajo. Este modelo irá evolucionando posteriormente mediante una cadena de transformaciones, incluyéndose en cada paso detalles más próximos a la implementación del sistema para, finalmente, generar de forma automática código que ayude al cumplimiento de los requisitos de seguridad elicitados como, por ejemplo, código que gestione las políticas de seguridad del sistema o los mecanismos de control de acceso sobre los activos de éste. Estos modelos serán explicados en detalle en el Capítulo 4.

3.4. Fase 3: análisis de modelos

En esta fase tomarán parte algunos de los modelos conceptuales obtenidos durante la *Fase 2*, los cuáles serán analizados con el objetivo de, en la medida de lo posible, detectar errores de diseño, avisar al modelador de situaciones que podrían ser anómalas para su inspección y, en definitiva, identificar mejoras que podrían ser realizadas en tiempo de modelado antes de que los errores se propaguen a fases posteriores del ciclo de desarrollo. Los elementos que toman parte en esta fase se muestran en la Figura 3.5. Concretamente, esta fase recibirá como entrada los modelos navegacionales y de usabilidad diseñados en la fase anterior junto con sus respectivos metamodelos. Tras la ejecución de las dos actividades de las que consta esta fase, se obtendrá información acerca de errores detectados en los modelos, métricas sobre ellos, problemas de consistencia o requisitos de usabilidad incumplidos. Toda esta información servirá como *feedback* para las fases de elicitación de requisitos y modelado del sistema y podrá ser utilizada para mejorar los artefactos creados en cada una de estas etapas. A continuación se detallan los objetivos la actividades mencionadas.

3.4.1. Actividad 3.1: evaluación de requisitos de usabilidad

Esta actividad recibirá los modelos navegacionales y los modelos de usabilidad del sistema, obtenidos en ambos casos durante la *Fase 2*, lo cuáles serán analizados, evaluándose de forma automática si los requisitos de usabilidad elicitados en la *Fase 1* y representados sobre los modelos de usabilidad en la *Fase 2* son realmente satisfechos por los modelos navegacionales. Como resultado de este proceso se obtendrá un informe que indique si estos modelos cumplen con los requisitos de usabilidad establecidos o si, por el contrario, existen errores como requisitos de usabilidad incumplidos o situaciones anómalas, como por ejemplo características de usabilidad que podrían ser mejoradas. Estas situaciones serán notificadas para su posterior inspección y, solución, si así se requiere. En el Capítulo 5 se detallará en profundidad esta actividad y los modelos y metamodelos implicados.

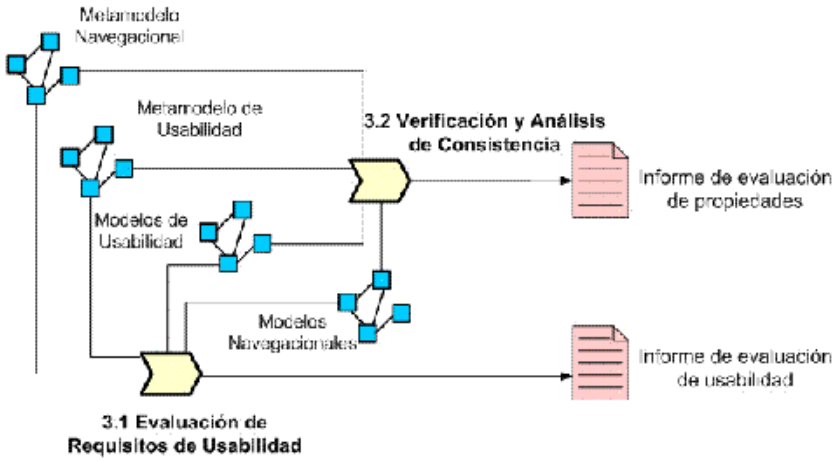

Figura 3.5: Fase 3: Análisis de modelos conceptuales

3.4.2. Actividad 3.2: verificación y análisis de consistencia

De nuevo el objetivo de esta etapa es analizar los modelos construidos en la *Fase 2*. Concretamente, en esta actividad se aplicarán técnicas de verificación de propiedades sobre los modelos navegacionales del sistema y se han establecido las bases para realizar tareas de análisis de consistencia entre estos modelos y los modelos de usabilidad creados en la *Fase 2* (esta será una línea de trabajo a abordar en el futuro).

Como se comentó en el Capítulo 2, el lenguaje Maude será usado para llevar a cabo estas tareas de análisis de modelos. Por ello, esta actividad necesitará tomar como entrada los modelos de navegación y usabilidad construidos en la fase anterior junto con sus respectivos metamodelos y transformarlos al lenguaje Maude para poder ser manipulados. Posteriormente el usuario podrá seleccionar los chequeos que desea realizar sobre los modelos a través de una herramienta que oculta los detalles del formalismo, que suelen ser ajenos a los modeladores, obteniendo como resultado un informe acerca de los errores y anomalías detectados. Los detalles relativos a esta actividad serán explicados en profundidad en el Capítulo 6.

3.5. Fase 4: mejora de artefactos

Esta fase simplemente resumiría el proceso de recoger el *feedback* obtenido de la fase previa y usarlo para solucionar los posibles problemas detectados. Este proceso puede implicar cambios tanto a nivel de requisitos (modificación de requisitos, eliminación de requisitos inconsistentes, etc.) como sobre los modelos conceptuales del sistema.

Capítulo 4

Metamodelado de Requisitos

4.1. Introducción

En este capítulo se explicarán las contribuciones aportadas por esta tesis doctoral en relación con el metamodelado de requisitos. Una vez dadas unas nociones básicas sobre metamodelado en la Sección 4.2, la Sección 4.3 mostrará los detalles de la revisión de la literatura llevada a cabo con el fin de identificar las propuestas existentes para el metamodelado de requisitos a partir de la que se ha definido un metamodelo propio que será mostrado en la Sección 4.4. Una vez definido este metamodelo, en la Sección 4.5 se explicará como éste ha sido integrado con un metamodelo de medición para dar soporte al concepto de requisitos medibles. La propuesta de metamodelo de requisitos medibles ha sido integrada en el ámbito de la Ingeniería Web tal y como se muestra en la Sección 4.5.3 y dotada de una notación visual que facilita su uso y que será mostrada en la Sección 4.5.4. Por último, la Sección 4.7 mostrará como el metamodelo de requisitos definido puede ser extendido para ofrecer soporte a la elicitación temprana de requisitos de seguridad.

4.2. Metamodelado de requisitos

Como se mencionó en capítulos previos, las propuestas de desarrollo dirigido por modelos están introduciendo un nuevo enfoque para la gestión de los requisitos del software que está basado en el uso de metamodelos de requisitos que definen formalmente los conceptos y relaciones implicados en el proceso de Ingeniería de Requisitos. De esta forma, y a diferencia de otras técnicas tradicionales para la recogida de requisitos como las plantillas o los casos de uso, los requisitos son también representados como modelos conformes a metamodelos bien definidos, lo que permite una mejor integración de los mismos en el proceso de desarrollo propuesto por los paradigmas *model driven*.

El uso de estos metamodelos ofrece varias ventajas [156]. Además de definir de forma precisa tanto las entidades como las relaciones que toman parte en el proceso de gestión de

requisitos, los metamodelos ofrecen una base formal sobre la que es posible construir de manera relativamente sencilla herramientas para manejar los modelos resultantes o para definir transformaciones automáticas desde estos modelos de requisitos a otros artefactos software como, por ejemplo, documentos de requisitos [157] o modelos conceptuales [158].

En los últimos años han aparecido diferentes propuestas para el metamodelado de requisitos pero, sin embargo, éstas presentan una gran disparidad en aspectos como el nivel de abstracción, número de conceptos considerados o semántica de los mismos. Como se mencionó en el Capítulo 2, esta falta de acuerdo en cuanto a los conceptos que deberían formar parte de un metamodelo de requisitos imposibilita la existencia de un metamodelo de referencia [7].

En el Capítulo 1 se estableció como uno de los objetivos de esta tesis la definición de un metamodelo de requisitos que, posteriormente, pudiera ser usado para su conexión con otros elementos como metamodelos de medición y metamodelos específicos para la consideración de requisitos de seguridad. Para su definición, la primera tarea llevada a cabo consistió en una revisión de la literatura que permitiera identificar las propuestas existentes para el metamodelado de requisitos. La comparativa realizada entre las diferentes propuestas estudiadas fue usada como base para la posterior definición de un metamodelo propio que aglutina los conceptos y mejores prácticas detectadas tras el proceso de revisión. En la siguiente sección se explican los detalles de esta revisión y una comparativa entre los metamodelos identificados.

Antes de ello, es importante resaltar que en la construcción de modelos de requisitos se suele distinguir entre modelos para la elicitación de *early requirements* (metas organizacionales) y *late requirements* (propiedades y capacidades del sistema) dependiendo de si estos modelos se centran en los objetivos de negocio o en los requisitos del sistema a ser construido [159]. En este caso nos centraremos en metamodelos de requisitos orientados a la elicitación de *late requirements*, que pueden ser complementados con modelos centrados en *early requirements* que, por ejemplo, pueden obtenerse a través de propuestas *goal oriented* como Tropos [160], KAOS [161] o NFR [162].

4.3. Comparativa entre metamodelos de requisitos

Con el objetivo de identificar el estado del arte con respecto a las propuestas para el metamodelado de requisitos se ha planteado un proceso de revisión sistemática de la literatura. Una revisión sistemática proporciona un medio para identificar, evaluar e interpretar la literatura relevante con respecto a una determinada área o cuestión de investigación [163]. Una revisión sistemática puede ser llevada a cabo con diferentes objetivos como sintetizar la evidencia existente en la literatura sobre una determinada cuestión (por ejemplo, un método, una técnica o cierta tecnología), para identificar y definir un marco de trabajo en el que situar nuevas actividades de investigación o para identificar carencias y aspectos a reforzar en un área de trabajo.

Para facilitar la planificación y ejecución de revisiones sistemáticas en Ingeniería del Software se han propuesto plantillas, guías y protocolos adaptados a este área y que

habían sido usados previamente en disciplinas científicas como la Medicina. Éste es el caso del protocolo de revisión propuesto por Kitchenham [163], el cuál seguiremos a lo largo de esta sección y que establece la definición de diferentes aspectos como el ámbito de la revisión, la formulación de una cuestión de investigación, la definición de criterios de inclusión y exclusión de los trabajos seleccionados y la extracción de información a partir de los resultados obtenidos. En las siguientes secciones mostraremos los detalles de este proceso de revisión, pudiendo encontrarse una descripción completa de todas las fases y actividades concernientes al mismo en [164].

4.3.1. Ámbito de la revisión y cuestión de investigación

Como se mencionó previamente, el objetivo de la revisión será obtener información sobre el estado del arte en el área de metamodelado de requisitos, identificando las propuestas surgidas en este ámbito y los conceptos cubiertos por cada una de ellas. La información obtenida será utilizada posteriormente para la definición de un metamodelo de requisitos que reúna los conceptos y mejores prácticas identificados durante el proceso de revisión. Siguiendo el protocolo definido por Kitchenham [163], para cumplir este objetivo se debe definir una cuestión de investigación que, en este caso, es la siguiente:

- *Q1: ¿cuáles son las propuestas existentes en la literatura para el metamodelado de requisitos?*

4.3.2. Proceso de búsqueda

Una vez definida la cuestión de investigación, es necesario definir un proceso para la búsqueda de información que ayude a su respuesta. En primer lugar, se debe seleccionar un conjunto de fuentes para la búsqueda de estudios que, en este caso, ha sido el formado por las siguientes:

- IEEE Digital Library (www.computer.org/portal/site/csdl/index.jsp)
- ACM Digital Library (portal.acm.org)
- Google Scholar (scholar.google.com)
- ScienceDirect (www.sciencedirect.com)
- CiteSeer^x (http://citeseerx.ist.psu.edu/)

Estas fuentes indexan artículos publicados en conferencias y revistas con calidad reconocida por la comunidad científica. Además, es importante destacar que fuentes como *Google Scholar* indexan a su vez a otras como *SpringerLink*, incluyéndose por tanto las actas de conferencias publicadas por esta editorial (como *Lecture Notes in Computer Science*, LNCS).

Todas las fuentes citadas ofrecen motores de búsqueda sobre los que se ha ejecutado una cadena de consulta formada por palabras clave adecuadas para la cuestión de investigación y que se ha definido de la siguiente forma:

- *(requirements AND (metamodeling OR metamodelling OR metamodel)) OR (metamodel of requirements)*

4.3.3. Criterios de inclusión y exclusión

La ejecución de la cadena de búsqueda sobre las fuentes comentadas ofrece múltiples resultados por ser los términos de la consulta comunes en el área de Ingeniería de Requisitos. En muchos casos, los trabajos están fuera del ámbito de la revisión y pueden ser descartados simplemente mediante la lectura del título y resumen de los mismos. En el caso de que esto no sea suficiente para determinar la relevancia o no del trabajo, se procede a la lectura de la introducción o el trabajo completo para evaluar si procede su inclusión.

En cuanto a los criterios de exclusión, se ha decidido considerar únicamente aquellos trabajos escritos en inglés y relacionados con el metamodelado de requisitos, por ser éste el propósito de la revisión.

4.3.4. Síntesis de resultados

Tras la aplicación del protocolo de revisión, se identificaron un total de 9 trabajos candidatos a ser considerados. El número de propuestas obtenidas no es muy elevado pero ello se explica porque el área de estudio es muy concreta y surgida de forma reciente. Siguiendo los criterios de exclusión mencionados en la sección anterior, tres de los nueve trabajos han sido descartados mientras que los otros seis se muestran en la Tabla 4.1. Como se explicará más adelante, la última columna de esta tabla indica los conceptos considerados por la propuesta de metamodelado de requisitos realizada en esta tesis y que ha sido denominada URM (*Unified Requirements Metamodel*).

Los trabajos no considerados finalmente han sido los de Escalona *et al.* [1, 120] y López *et. al* [165]. En el caso de [120], ha sido descartado por ser un metamodelo específicamente centrado en sistemas basados en web que sigue una aproximación demasiado centrada en el diseño de estos sistemas tal y como se desprende de la inclusión en el metamodelo de conceptos como nodos de navegación o elementos de búsqueda. En [1] se muestra una versión ligeramente revisada del metamodelo presentado en [120], pero éste sigue la misma filosofía centrada en el diseño del sistema. En el caso de [165], ha sido descartado por ser un metamodelo orientado a la integración de diferentes tipos de diagramas más que al metamodelado de requisitos.

Pasemos a continuación a comparar los trabajos seleccionados. Como se observa en la Tabla 4.1, los conceptos presentes en cada metamodelo varían y, además, no se suelen ofrecer definiciones precisas sobre la semántica de los conceptos considerados, hecho que

	Bolchini [118]	COMET [166]	Fernández [167]	Goknil [7]	REMM [157]	SME [168]	URM [25]
Caso de test				✓	✓	✓	✓
Catálogo					✓		✓
Descripción Adicional / Caso de uso		✓	✓	✓			
Escenario		✓	✓				✓
Fuente					✓		✓
Glosario				✓	✓		✓
Meta	✓	✓	✓				✓
Participante (*stakeholder*)	✓	✓	✓(Actor)	✓	✓	✓	✓
Requisito	✓	✓	✓	✓	✓	✓	✓
Requisito Funcional		✓	✓			✓	✓
Requisito No Funcional		✓	✓				✓
Término			✓	✓	✓		✓
Traza				✓	✓		

Tabla 4.1: Comparativa de metamodelos de requisitos surgidos en el ámbito de MDE

abre dudas acerca de si diferentes metamodelos entienden lo mismo sobre conceptos con el mismo nombre. El concepto de `requisito` (entendido generalmente como una condición o capacidad que el sistema debe poseer para satisfacer un contrato, especificación, etc.) es el único presente en todos los metamodelos. En los metamodelos estudiados no existe consenso sobre si se deben proporcionar categorías para los requisitos, siendo la clasificación más habitual, en caso de existir, la que los divide en requisitos funcionales (que indican lo que el sistema debe hacer) y requisitos no funcionales (que indican restricciones acerca de cómo el sistema debe llevar a cabo su tarea y están relacionados con aspectos como, por ejemplo, seguridad, fiabilidad, usabilidad, etc.).

Sin embargo, existe mucha disparidad en cuanto al resto de elementos considerados. Por ejemplo, de los seis metamodelos identificados, tres de ellos incluyen el concepto de `meta` (*goal*), usado para modelar un objetivo de alto nivel [118, 166, 167]. En otros casos como en REMM [157] se metamodela el concepto de reutilización de requisitos a través del uso de `catálogos` de requisitos que sirven para agrupar un conjunto de requisitos extraídos de una o más `fuentes` (por ejemplo, leyes, políticas organizacionales, guías, etc.). Los requisitos agrupados en un catálogo pueden ser reutilizados en todos aquellos proyectos donde las fuentes correspondientes sean aplicables [122, 169].

El hecho de que los requisitos suelan ser descritos de forma textual ha motivado la consideración de otros elementos en su metamodelado. Por un lado, tal y como ocurre en los metamodelos descritos en [7, 166, 167], se ha considerado el concepto de `descripción adicional` que, para ayudar a entender el requisito, ofrece información más detallada

acerca del mismo usando, por ejemplo, texto o algún otro artefacto software como casos de uso de UML. También, sabida la ambigüedad propia del lenguaje natural, se propone que la descripción de los requisitos se haga usando una serie de `términos` bien definidos que pueden agruparse para formar `glosarios` que ayudan en la tarea de definir y entender los requisitos.

Dos de los seis metamodelos incluyen el concepto de trazabilidad, permitiendo la definición de relaciones entre dos o más requisitos [7, 157]. Estas relaciones pueden ser de distintos tipos (padre/hijo, refinamiento, conflicto, etc.) y se reflejan a través del concepto de `traza`.

También es importante resaltar que varias propuestas [7, 157, 168] han prestado atención a la validación de requisitos, resaltando así la necesidad de chequear que el sistema o software implementado cumple con los requisitos definidos. Con este propósito, estos metamodelos incluyen el concepto de `caso de test`. Sin embargo, no se han proporcionado guías que ayuden a determinar la forma en la que esta validación debería ser definida para que el proceso de validación fuera repetible, sistemático y, si fuera posible, automático.

Además de los conceptos mencionados, durante la comparación aparecieron otros conceptos más específicos en los que existe aún menos consenso como tareas, puntos de vista, restricciones de diseño o características propias de dominios concretos que deben dejarse fuera si se pretenden definir metamodelos de propósito general.

Otro dato importante de la comparativa reside en que sólo algunos metamodelos como [157] ofrecen algún tipo de soporte automático para tratar con los conceptos propuestos. En cuanto a la definición de transformaciones para la generación de otros artefactos software usando como base el metamodelo de requisitos, [157] propone transformaciones modelo a texto que permiten obtener los requisitos especificados organizados de acuerdo con el estándar de especificación de requisitos IEEE-830 [170]. Por otro lado, también se observó que los metamodelos identificados tampoco suelen ofrecer una sintaxis concreta (notación) para tratar con los conceptos que forman el metamodelo. Estos aspectos son mostrados en la Tabla 4.2, explicándose en las siguientes secciones la forma en la que se trata con ellos en el marco de esta tesis doctoral.

Los datos extraídos de esta revisión de la literatura han servido para identificar un conjunto de conceptos que debería formar parte de un metamodelo de requisitos y nos han permitido realizar una propuesta propia de metamodelo de requisitos (URM) que es explicada en la siguiente sección.

4.4. URM: una propuesta para el metamodelado de requisitos

Dada la falta de acuerdo con respecto a los conceptos que deben formar parte de un metamodelo de requisitos, se ha propuesto la definición de un metamodelo que aglutine los conceptos que habitualmente aparecen en las propuestas existentes así como las mejores

	Bolchini [118]	COMET [166]	Fernández [167]	Goknil [7]	REMM [157]	SME [168]	URM [25]
Herramienta de soporte				✓	✓		✓
Notación (sintaxis concreta)					✓		✓
Conexión con métricas							✓
Extensiones para tipos específicos de requisitos							✓

Tabla 4.2: Otras características consideradas en la comparativa de metamodelos

prácticas detectadas en ellas como el soporte para la reutilización de requisitos, para la trazabilidad de los requisitos tanto hacia atrás, alineándolos con objetivos de negocio, como hacia adelante, conectándolos con métodos de validación o la diferenciación entre requisitos funcionales y no funcionales. Como veremos a lo largo de este capítulo, estas dos últimas características servirán como punto de extensión para conectar al metamodelo propuesto extensiones específicas para el modelado de métodos de validación o el modelado de requisitos no funcionales de tipos específicos, como los relacionados con la seguridad del sistema. Además, el metamodelo se ha mantenido simple para permitir su rápida instanciación e integración en metodologías de desarrollo dirigido por modelos.

URM, que es mostrado en la Figura 4.1, además de servir como mecanismo para la elicitación de requisitos, ha sido dotado de un soporte automático que será explicado en la Sección 4.6 a través de un ejemplo. Durante la elaboración de esta tesis doctoral, el metamodelo ha sido refinado y ampliado. Inicialmente, surgió como un metamodelo específico para Ingeniería Web [23] pero posteriormente se amplió a un metamodelo de propósito general que también podía seguir siendo usado en este ámbito [22]. Además, como se ha comentado y se irá detallando a lo largo de las siguientes secciones, el metamodelo definido ha sido usado como base para su integración con un metamodelo de medición [24, 25], habiendo sido ambos integrados en el ámbito de la Ingeniería Web [27].

De la misma forma que en los diferentes metamodelos estudiados, el principal elemento del metamodelo de la Figura 4.1 es el de `requisito`, que es descrito utilizando un conjunto de atributos (omitidos en la Figura 4.1) formado por su identificador, su tipo, su prioridad y una descripción textual del mismo. Los requisitos pueden ser clasificados en requisitos funcionales y no funcionales. Esta clasificación resulta útil ya que mientras que los requisitos funcionales suelen ser validados a través de `casos de test`, los requisitos no funcionales están relacionados con `escenarios de calidad` que cuantifican y contextualizan un determinado aspecto de calidad [171].

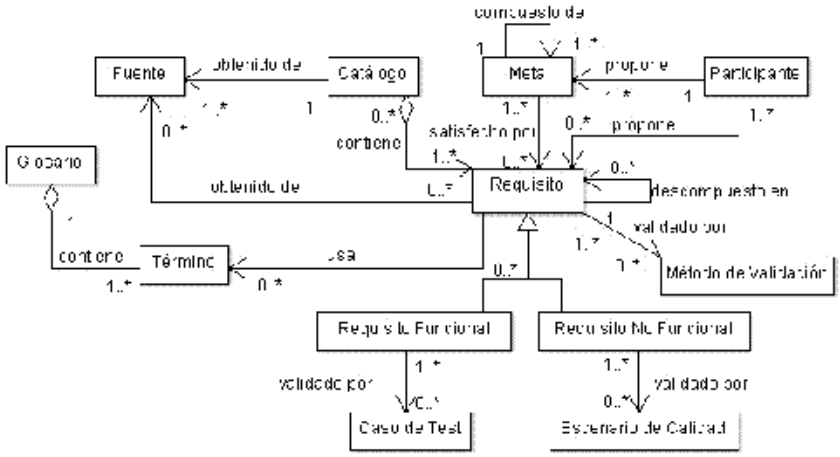

Figura 4.1: URM: una propuesta para el metamodelo de requisitos

Debido a la necesidad de trazabilidad entre requisitos y objetivos de negocio que podrían estar recogidos en modelos organizacionales, cada requisito ha sido conectado con el conjunto de metas que contribuye a satisfacer. Cada `meta` sirve para capturar un objetivo de alto nivel propuesto por uno o más `participantes` (también denominados *stakeholders*) y, a su vez, puede ser descompuesta en metas más concretas.

Como se observa en la Figura 4.1, URM también incluye el concepto de `catálogo`, que agrupa un conjunto de requisitos relacionados entre sí, favoreciendo de este modo la reutilización de requisitos y permitiendo acelerar el proceso de instanciación del metamodelo.

Con el doble objetivo de evitar en lo posible la ambigüedad inherente al lenguaje natural y de ayudar a los desarrolladores a entender los requisitos (generalmente expresados en términos que únicamente son familiares para los expertos en el dominio), se han introducido los conceptos de `término` (que representa a un concepto junto con una definición precisa del mismo) y `glosario` (que agrupa un conjunto de términos). Además, el metamodelo incluye la metaclase `método de validación`, la cuál, como veremos en más detalle en la siguiente sección, nos servirá para conectar los requisitos extraídos con métricas que ayuden a validar su cumplimiento.

Una vez mostrado el metamodelo de requisitos de la Figura 4.1, veamos como puede ser conectado con un metamodelo de medición con el objetivo de dar soporte al concepto de requisitos cuantificables.

4.5. Conectando metamodelos de requisitos y medición

4.5.1. Introducción

La definición explícita de los requisitos de un sistema, aunque necesaria, puede no ser suficiente para garantizar la calidad del producto resultante a partir de ellos [172]. Muchos

factores pueden influir en la calidad del sistema desarrollado (por ejemplo, las decisiones tomadas durante su proceso de construcción) y es por ello que surge la necesidad de dotar a las metodologías de desarrollo con técnicas que ayuden a validar la calidad de los productos obtenidos [173]. Mientras que en el caso de requisitos funcionales la tarea de validación puede resultar sencilla (en el sentido de que el ingeniero software puede determinar si una funcionalidad requerida es realmente cumplida o no por el sistema), la validación de requisitos no funcionales resulta más compleja necesitándose para ello un espacio de calidad objetivo, bien definido y compartido entre los *stakeholders* del sistema. Para conseguir la mencionada objetividad se hace necesario que los requisitos sean cuantificables. Un método extendido para cuantificar requisitos no funcionales consiste en la definición de medidas asociadas a ellos que convierten a éstos en requisitos cuantificables [49]. Esta práctica ha sido incluida en diferentes procesos de desarrollo software (por ejemplo, [174]) en los cuáles ha sido probada su contribución al éxito de los proyectos desarrollados.

Para conseguir que estas medidas sean soportadas e incluidas en el ámbito del desarrollo dirigido por modelos y para permitir la automatización de las mismas, es necesario que la definición de medidas sea soportada por metamodelos [175]. Un metamodelo de referencia en este sentido es el denominado *Software Measurement Metamodel* (SMM [2]), que está basado en una ontología de medición (*Software Measurement Ontology*, SMO) que incluye los conceptos necesarios en este proceso como medidas, tipos de medidas, conceptos medibles, escalas, etc.

Sin embargo, hasta donde llega nuestro conocimiento, no se ha trabajado en la conexión entre estos metamodelos de medición y los metamodelos de requisitos existentes, siendo éste otro de los objetivos de esta tesis doctoral. La conexión entre ambos tipos de metamodelos supondría beneficios en dos sentidos. Por un lado, la inclusión de medidas precisas y bien definidas junto con requisitos incrementaría la precisión y repetibilidad del proceso de validación de requisitos. Por otro lado, la definición de requisitos y medidas en forma de modelos ayudaría a la automatización del proceso de medición, acelerándose así el proceso de validación de requisitos. Además, las propuestas MDD proporcionan modelos sobre los que sería posible realizar estas medidas permitiéndose por tanto una detección y corrección temprana de problemas que disminuye drásticamente los costes del proyecto [176].

Por tanto, con el fin de soportar el concepto de requisito cuantificable mediante medidas, el metamodelo de requisitos mostrado en la Figura 4.1 ha sido complementado con una extensión para la medición de requisitos a la que hemos denominado SMM−, utilizándose el SMM como base para ello.

4.5.2. Adaptando SMM para el soporte de requisitos medibles

El *Software Measurement Metamodel* presentando en García *et al.* [2] es mostrado en la Figura 4.2. Este metamodelo está basado en una ontología de medición subyacente realizada con el propósito de proporcionar una armonización entre los diferentes estándares y propuestas para la medición de software junto con una manera precisa de definir

medidas. Sin embargo, su idiosincrasia ha requerido realizar algunos ajustes para nuestro propósito de enlazarlo con el metamodelo de requisitos mostrado en la Figura 4.1.

Por un lado, algunos de los conceptos en SMM estaban definidos a diferentes niveles de abstracción, algo que no era adecuado para nuestro propósito de modelado. Otros conceptos simplemente se solapaban parcialmente con los conceptos recogidos en el metamodelo de requisitos, que en este sentido proporcionaba constructores más precisos. Los ajustes realizados se resumen de la siguiente forma:

- SMM incluye *Conceptos Medibles*, los cuáles componen un *Modelo de Calidad*. En URM este concepto es equivalente al de requisito medible, el cuál puede ser extraído de una determinada *Fuente* (que en realidad puede representar no sólo un modelo de calidad, sino también otros tipos de fuente, tales como estándares). Por tanto, hemos dejado fuera de nuestra propuesta ambos conceptos de SMM.
- Debido a su base ontológica, SMM incluye un paquete denominado *Acción de Medir*, que incluye elementos como *Medición* (la acción de aplicar una medida sobre una entidad dada) o *Resultado de la Medición*. Este paquete no es útil a nivel de modelado, y por tanto queda fuera de nuestra propuesta. Del mismo modo, el concepto *Entidad* hace referencia al artefacto concreto sobre el que la medida es aplicada. Dado que la entidad concreta puede ser vista como una ocurrencia del concepto *Categoría de Entidad*, dicho concepto de *Entidad* ha quedado también fuera de nuestra propuesta.
- Los *Indicadores* en SMM están relacionados con el concepto de *Necesidad de Información*, generalmente definido como "entendimiento necesario para gestionar objetivos, metas, riesgos, y problemas" [2]. Sin embargo, esta definición sugiere mucho más que una simple descripción textual. De hecho, la definición de metas, riesgos, escenarios de calidad, etc. ha constituido el ámbito tradicional de los metamodelos de Ingeniería de Requisitos. Por tanto, en la propuesta realizada, el URM sustituye, completa y elimina la ambigüedad del concepto *Necesidad de Información* que estaba presente en el SMM.

El resultado de estas consideraciones es lo que se ha denominado SMM– y se muestra en la Figura 4.3. Como puede observarse, en el metamodelo SMM– los elementos se organizan alrededor del concepto de `medida`. Las medidas puede ser subdivididas en `medidas base` (directamente calculadas por medio de un método de medición), `medidas derivadas` (obtenidas a partir de una o más medidas base y calculadas a través de una función de medición) o `indicadores`. Los indicadores, calculados por medio de `modelos de análisis`, pueden estar compuestos tanto de medidas base como derivadas y tienen asociado un `criterio de decisión`. Estos criterios de decisión convierten los resultados de la medición en respuestas para cierta necesidad de información. En otras palabras, en nuestra propuesta los indicadores sirven para validar requisitos no funcionales. Además, para especificar de forma completa una medida deben proporcionarse elementos como su `escala`, la `unidad de medida` y el `tipo de escala`. Además, las medidas son aplicadas sobre cierto `tipo de entidad`, que en este caso se corresponde con cualquier modelo

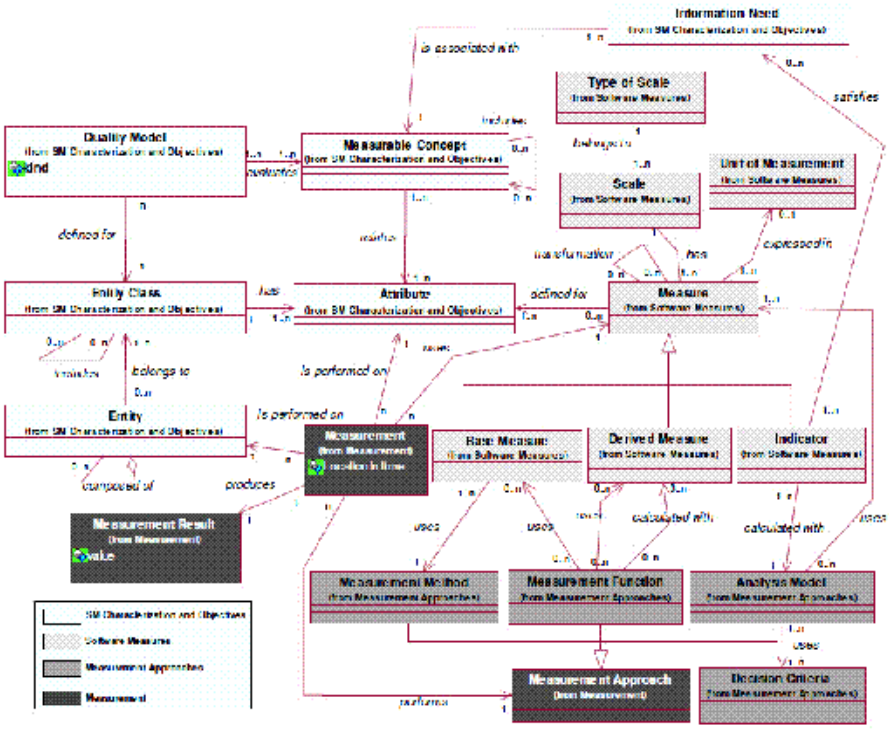

Figura 4.2: Metamodelo para la medición de Software (SMM [2])

incluido en el proceso de desarrollo. Los modelos tienen determinados atributos (como tamaño, complejidad, longitud, acoplamiento y cohesión [177]) y las medidas pueden ser formalmente relacionadas con estos atributos.

Para conectar los metamodelos de las Figuras 4.1 y 4.3 se ha usado como punto de unión por un lado (i) la metaclase ***Indicador*** (que es un subtipo de ***Medida***) y por otro (ii) la metaclase ***Método de Validación*** que se ha añadido a la Figura 4.3 para plasmar esta conexión. Es importante resaltar que aunque, idealmente, todos los requisitos deberían ser medibles, en la práctica las restricciones de tiempo y coste hacen que esta alternativa sea inviable [49]. Por ello, nuestra propuesta, al igual que otras como la de Larman [174], recomienda que con el objetivo de no sobrecargar el proceso de modelado únicamente aquellas medidas que realmente vayan a ser tenidas en cuenta durante el proceso de testeo sean modeladas.

A la propuesta resultante de la unión entre URM y SMM– se la ha denominado MRM (*Measurable Requirements Metamodel*). La complementariedad de ambos metamodelos en MRM se traduce en una capacidad expresiva mayor que la proporcionada por otros metamodelos. Por un lado, MRM amplía la capacidad expresiva de los metamodelos de requisitos tradicionales, al permitir definir requisitos medibles. Por otro lado, MRM también mejora la capacidad para expresar medidas del propio SMM, ya que en MRM es posible expresar de forma mucho más sistemática no sólo el propósito (requisitos y, en última instancia, metas de negocio a las que se asocian) de las medidas, sino también

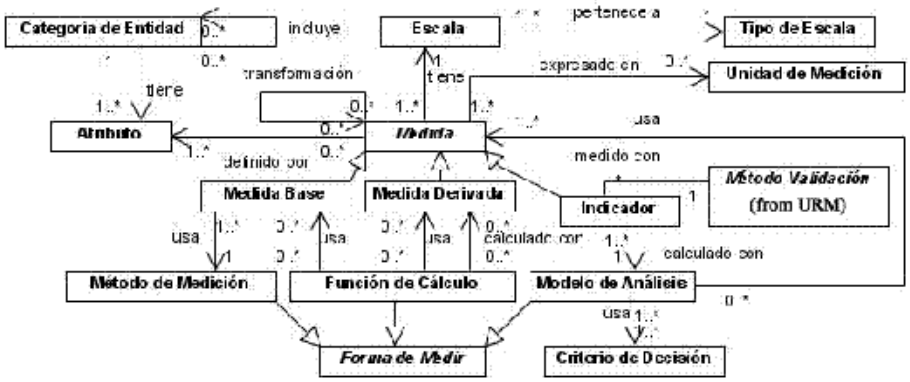

Figura 4.3: SMM–: adaptando SMM para su conexión con URM

su contexto de aplicación. En MRM, este contexto incluye escenarios de calidad con condiciones bajo las que las medidas deben ser aplicadas. Toda esta información es de suma importancia para la definición de medidas, ya que afecta a la aceptabilidad de su resultado. Por ejemplo, es diferente aplicar una medida de rendimiento sobre un sistema con una carga de 10 usuarios simultáneos (escenario de calidad 1) que aplicar la misma medida con 5000 usuarios simultáneos (escenario de calidad 2). Dichos escenarios de calidad no eran tenidos en cuenta de manera explícita en el SMM.

Una vez definido MRM, el siguiente paso en esta tesis doctoral fue su integración en el ámbito de la Ingeniería Web. La forma en la que esta integración ha sido llevada a cabo se explica en detalle en la Sección 4.5.3.

Por otro lado, la definición de modelos a partir de MRM puede realizarse de al menos dos formas. Una opción para ello sería la instanciación directa de las metaclases de MRM, construyendo por tanto modelos de objetos (ver Figura 4.9). Otra opción consistiría en la definición previa de una sintaxis concreta (nueva iconografía) asociada a las metaclases de MRM utilizándose estos iconos para la definición de modelos. Sin embargo, ambas posibilidades tienen inconvenientes. Por un lado, la experiencia ha demostrado que los modelos de objetos no suelen presentar buenas características de escalabilidad, por lo que su manejo tiende a complicarse cuando el sistema a modelar incrementa su complejidad [178]. Por otro lado, la definición de una nueva iconografía hace surgir varias cuestiones. Por ejemplo, los usuarios de MRM tendrían que familiarizarse con esta nueva notación y, además, habría que intentar descubrir qué iconos serían más adecuados para cada concepto, con objeto de facilitar a los modeladores la tarea de crear, recordar y utilizar modelos conformes con MRM.

Una alternativa para paliar estos problemas consitiría en la reutilización de alguna notación existente y conocida en la comunidad de Ingeniería de Requisitos, evitando de este modo la necesidad de aprender una nueva notación. Finalmente, se optó por esta última posibilidad aunque, como se comentará en la Sección 7.3, ya se ha trabajado y, sobre todo, se abordará como trabajo futuro, la realización de una serie de experimentos con usuarios que permitan descubrir una notación con la que dotar a los elementos de MRM. Para ello, MRM ha sido complementado con una propuesta de perfil UML que

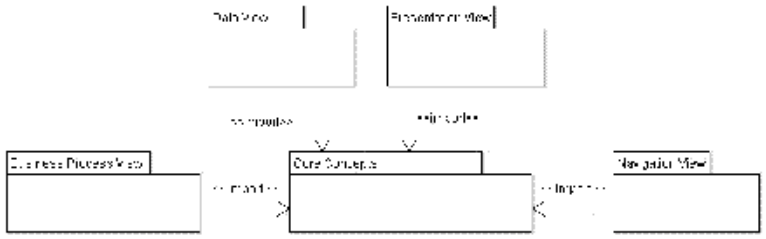

Figura 4.4: Estructura de paquetes en el *Common WE Metamodel*

disminuye la complejidad cognitiva de los modelos, al mismo tiempo que preserva su portabilidad y reusa una notación existente para el modelado de requisitos. Los detalles de este perfil serán mostrados en la Sección 4.5.4.

4.5.3. Integrando MRM en el ámbito de la Ingeniería Web

En esta sección se mostrará como MRM puede ser integrado en el ámbito de la Ingeniería Web. Como se mencionó en la Sección 2.4.1, la gestión adecuada de los requisitos es una de las líneas de trabajo abiertas en la disciplina de Ingeniería Web [37]. Las metodologías de desarrollo surgidas en este ámbito suelen estar principalmente centradas en el diseño del sistema, dejándose a un lado la gestión de requisitos, que ha sido considerada sólo en algunos trabajos [1, 120].

En este área resulta especialmente significativa la ausencia de conceptos relacionados con la gestión de requisitos en el *Common Web Engineering Metamodel* [55, 56], una iniciativa desarrollada recientemente con el objetivo de la armonización de los conceptos manejados en esta disciplina. En la Figura 4.4 se muestra una visión general de la estructura de paquetes que, hasta el momento, han sido propuestos en este metamodelo común. Como puede observarse, en la Figura 4.4 aparecen conceptos relacionados, por ejemplo, con la navegación o presentación del sistema pero no con la gestión de los requisitos del mismo. Por ello, se propone complementar esta estructura con un nuevo paquete que introducirá conceptos relacionados con requisitos.

Como se muestra en la Figura 4.4, el *Common Web Engineering Metamodel* está originalmente compuesto por cinco paquetes. Uno de ellos, denominado *Core Concepts,* aglutina una serie de conceptos básicos (por ejemplo, se incluye el concepto de modelo junto con los elementos que lo forman y las asociaciones que pueden establecerse entre ellos) mientras que el resto de paquetes heredan de éste y se centran en los conceptos relacionados con las vistas de datos, navegación, presentación y procesos de negocio.

La extensión que proponemos para este metamodelo común se muestra en la Figura 4.5. La propuesta ha sido denominada WE-RMExt (*Web Engineering Requirements Metamodel Extension*) y tiene el propósito de incrementar la importancia que la Ingeniería de Requisitos desempeña en Ingeniería Web. Siguiendo la filosofía de organización en paquetes propuesta, la extensión realizada ha sido encapsulada en un nuevo paquete, al que se ha denominado *vista de requisitos medibles* (*Measurable Requirements View* en la

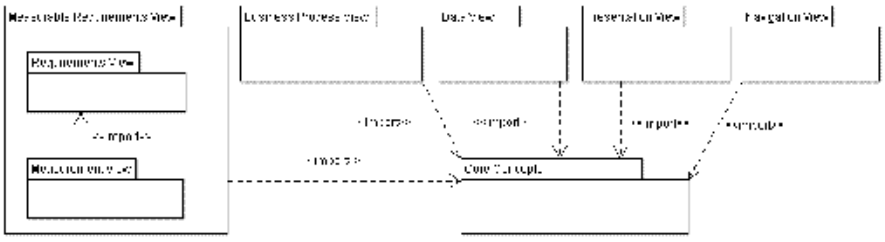

Figura 4.5: Integrando requisitos y medidas en el *Common WE Metamodel*

Figura 4.5). A su vez, este paquete está dividido en dos subpaquetes, uno para la vista de requisitos y otro para la vista de medición, incluyendo cada uno de ellos los conceptos recogidos por los metamodelos de requisitos y medición presentados en las Figuras 4.1 y 4.3.

La Figura 4.6 muestra, de forma simplificada, la forma en la que se ha realizado la integración de esta propuesta. Esta nueva vista de requisitos medibles reutiliza los elementos definidos en el paquete *Core Concepts* (ver Figura 4.4) siguiendo una estrategia similar a la del resto de paquetes. En la Figura 4.6, el nuevo modelo de requisitos (denominado `CM_ReqModel`), el cuál está compuesto por un conjunto de elementos (`CM_ReqElement`), extiende de la metaclase `CM_Model` existente en el paquete *Core Concepts*. Los conceptos que componen el metamodelo de requisitos de la Figura 4.1 (como requisito, meta, etc.) extienden al concepto `CM_ReqUnit` mientras que las relaciones entre los conceptos de la Figura 4.1 se han modelado a través del elemento `CM_ReqAssoc` que directamente extienden del concepto `CM_Relationship` de *Core Concepts*. Para facilitar la comprensión de la extensión, no todas las metaclases de la vista de requisitos han sido mostradas en la Figura 4.6. En el caso de la vista de medición se ha seguido la misma estrategia. Para ello, se ha definido un nuevo modelo de medición (`CM_MeasModel`) y los diferentes elementos que componen el metamodelo de medición y las asociaciones entre ellos (ver Figura 4.3) se modelan mediante los elementos `CM_MeasElement` y `CM_MeasAssoc`.

Desde nuestro punto de vista, la integración de WE-RMExt en el Common Metamodel contribuye a aproximar los requisitos del sistema con el diseño conceptual del mismo, aspecto en el que generalmente están centradas las diferentes metodologías surgidas en Ingeniería Web. Las vistas de requisitos y medición proporcionan un mecanismo para obtener requisitos y conectarlos con métodos de validación de forma sencilla, algo importante en una disciplina en la que el punto de partida (*baseline*) de los proyectos debe ser obtenido de forma rápida debido a la habitual existencia de tiempos cortos para el desarrollo de los proyectos [13]. Por otro lado, el hecho de que esta extensión incluya los conceptos de meta y participante (*stakeholder*) en lugar de únicamente el de usuario web contribuye a la conexión entre requisitos y objetivos de negocio. Esta integración puede ayudar a reducir el numero de fallos detectados en sistemas web con el objetivo final de incrementar la calidad de los sistemas desarrollados y la satisfacción de sus usuarios.

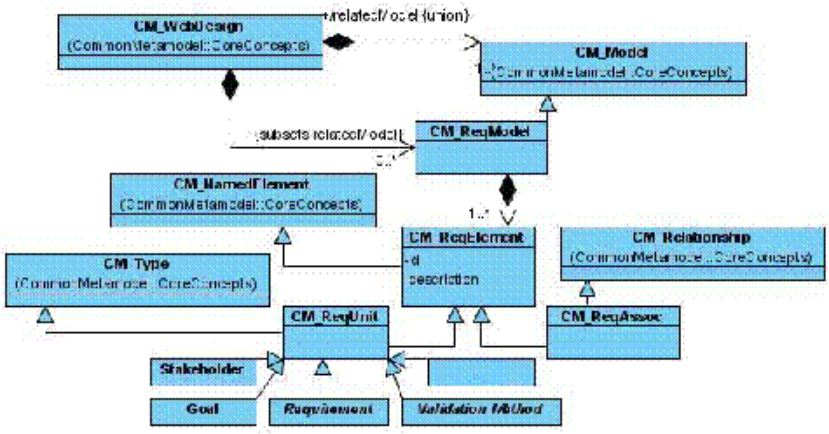

Figura 4.6: Extensión del *Common Metamodel* para la vista de requisitos

4.5.4. Una notación visual para MRM

Incluso con la ayuda de un metamodelo como MRM, conseguir modelos que sean al mismo tiempo concisos y completos no es una tarea sencilla si no se dispone de notaciones adecuadas. Además, la expresión visual de los modelos constituye un paso fundamental para su uso ya que si se dispone de los mecanismos visuales adecuados, se contribuye a mejorar la usabilidad y utilidad de los mismos como artefactos para el desarrollo de software [179].

Como puede observarse en la Figura 4.9, la definición de diagramas de objetos en los que se representan instancias de las metaclases que componen MRM no es suficiente para hacer uso eficiente de esta propuesta [180]. Por ello, MRM viene acompañado por una notación visual que facilita su instanciación y que se explica en las siguientes secciones. Entre los requisitos que se tuvieron en cuenta para definirla se consideró que la nueva notación resultase familiar para la comunidad de Ingeniería de Requisitos, que fuera capaz de cubrir todos los conceptos incluidos en MRM y que permitiera crear modelos más sencillos que los que se obtenían de la mera instanciación de MRM mediante modelos de objetos. Dadas estas consideraciones, se optó por la reutilización de una notación existente a través del uso de un perfil UML.

Aunque en la disciplina de Ingeniería de Requisitos se han propuesto muchas notaciones, el marco *i** ha ido imponiéndose en los últimos años como una de las más extendidas sobre todo en el ámbito de los enfoques orientados a metas. Dado que existen varias versiones de esta notación [181], para nuestro trabajo hemos seleccionado la propuesta URN (*User Requirements Notation* [182]), que ofrece un lenguaje de modelado denominado GRL (*Goal-oriented Requirements Language*) que permite la definición de conceptos como metas y requisitos no funcionales. Sin embargo, GRL no proporciona los conceptos necesarios para tratar con requisitos medibles, por lo que se hace necesario extender GRL para dar soporte a este concepto. Para proporcionar esta extensión, y dado que la notación preferiblemente debe ser modular [183], se ha decidido utilizar un mecanismo bien conocido como son los perfiles UML. El resultado es un perfil para MRM que es-

tá basado en un perfil preexistente para GRL [184] y que extiende éste con conceptos de medición. Este perfil permitirá el uso de una notación bien conocida como *i** para expresar un concepto no presente en este marco como es el de requisitos medibles.

4.5.4.1. Visión general del *mapping* entre GRL y UML

Cuando se prepara cualquier lenguaje para acoger a otro (en nuestro caso, GRL para dar soporte a MRM), es necesario que éste incluya algún tipo de punto de extensión a partir del que sea posible realizar la adaptación del nuevo lenguaje. El mecanismo requerido puede ser tan simple como la capacidad para anotar los elementos de modelado con etiquetas, notas o atributos. Por ejemplo, GRL incluye atributos de descripción en sus metaclases que pueden ser etiquetados para acomodar los conceptos de MRM. El inconveniente de esta estrategia es que los nuevos conceptos carecen de iconos visuales. Como otra alternativa, las herramientas de soporte a GRL proporcionan notas que pueden ser adjuntadas a los elementos de modelado para mostrar esta extensión. Sin embargo, el uso de notas puede resultar en diagramas sobrecargados que se vuelven más complicados de entender. Por tanto, parece que GRL por si solo no ofrece mecanismos adecuados para acoger a MRM.

Los problemas mencionados pueden ser resueltos de forma sencilla si se usa el lenguaje UML para dar soporte a GRL. Mientras que GRL no está originariamente concebido para acoger extensiones, UML ofrece soporte para la definición de perfiles, un mecanismo formal que permite extender tanto la sintaxis como la semántica de UML y que permite adaptarlo para ofrecer soporte a otros conceptos. Los perfiles están articulados alrededor de estereotipos asociados a las metaclases UML, cuyas definiciones etiquetadas permiten añadir atributos a dichas metaclases. Junto con esta extensión semántica (como parte de la sintaxis abstracta de UML), los estereotipos permiten definir notaciones (sintaxis concreta), como iconografías, para adaptar la notación estándar de UML a la notación específica del dominio. Los estereotipos pueden además venir acompañados de restricciones definidas en OCL (*Object Constraint Language* [185]), que son usadas comúnmente para formalizar la sintaxis abstracta de dominio que capturan estos perfiles. Usando este mecanismo se pueden superar las limitaciones en cuanto a capacidades de extensión ofrecidas por GRL. Además, el uso de UML para dar soporte a MRM permite emplear estos modelos como un artefacto adicional en procesos de desarrollo basados en el uso de UML y sus herramientas de soporte.

Así, para realizar esta extensión se utilizó como punto de partida el perfil UML para GRL propuesto por Mazón *et al.* [184] y que se puede observar en la parte superior de la Figura 4.7. El perfil está basado en las metaclases del metamodelo UML usadas comúnmente para definir perfiles como *Package*, *Association*, *AssociationClass* y *Class*. En la Figura 4.7 puede observarse como estas metaclases han sido adaptadas para tratar con la semántica de GRL considerando aspectos como dependencias estratégicas (*SD*), actores (*iActor*), los diferentes tipos de relaciones intencionales (*MeansEnd*, *Decomposition*, *Contribution*, *Correlation*, y *iDependency*) y el conjunto de elementos intencionales (*Belief*, *Goal*, *Task*, *Resource*, y *Softgoal*). Es importante hacer notar que este perfil pro-

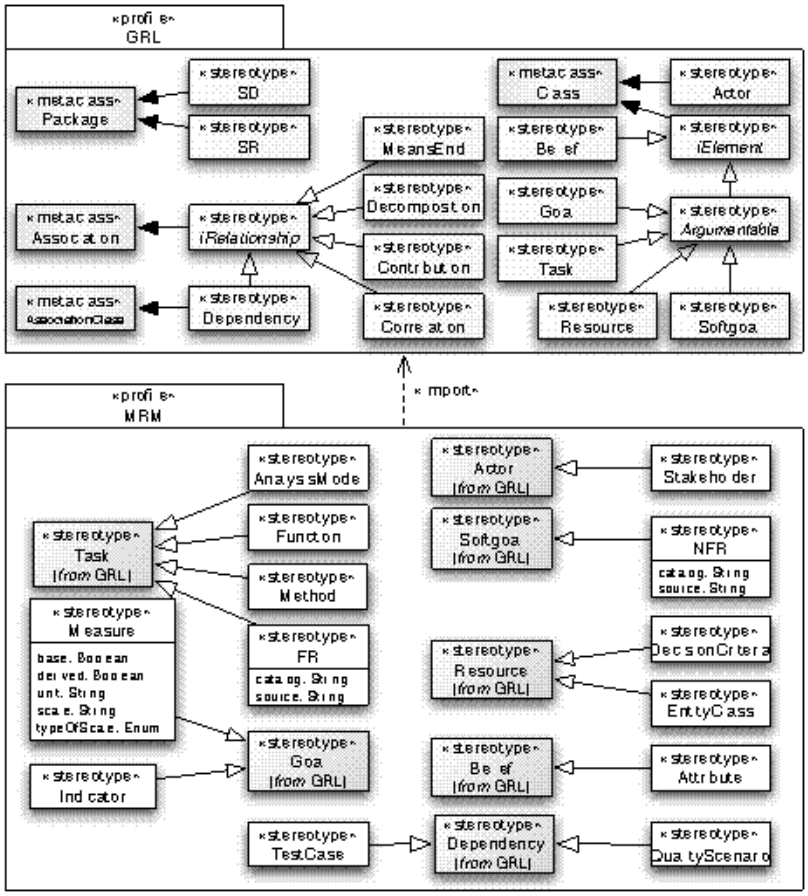

Figura 4.7: Arquitectura del perfil de soporte a MRM

porciona a los modeladores tanto la semántica de los elementos de GRL como su notación en UML.

4.5.4.2. Un perfil UML para visualizar MRM

En la Tabla 4.3 se muestran los nuevos estereotipos definidos (los cuáles utilizan los nombres de las metaclases de MRM), junto con los elementos del perfil para GRL que sirven como base para su extensión (ver la parte inferior de la Figura 4.7). Esta especialización contrasta con la relación UML de extensión que fue usada entre las metaclases y estereotipos definidos en el perfil GRL original. La razón de ello es que mientras que en el perfil GRL los elementos extendidos fueron metaclases UML, en el perfil para MRM se están extendiendo estereotipos con el propósito de adaptar el perfil existente a los nuevos conceptos de MRM. Esta adaptación es conseguida mediante la definición de jerarquías de herencia entre clasificadores de UML por medio de relaciones de generalización.

Correspondencia entre metaclases La extensión de MRM presentada en la Figura 4.7 se deriva de la correspondencia notacional mostrada en la Tabla 4.3. Lógicamente, las características de la notación elegida influyen en la efectividad de los diagramas construidos e incluso pequeños cambios en la notación pueden tener un impacto importante sobre la facilidad de entendimiento de los diagramas [186]. Por ello, para seleccionar esta notación se han seguido una serie de principios diagramáticos que intentan favorecer la efectividad de los modelos [178]. Veamos a continuación estos principios:

- **Discriminación perceptual**: en la notación propuesta, diferentes formas han sido asociadas con diferentes conceptos para facilitar el reconocimiento de los mismos (por ejemplo, pueden compararse los diagramas de las Figuras 4.9 y 4.10).
- **Configuración perceptual**: la notación propuesta está basada en el uso de iconos. Estos iconos hacen que los diagramas sean más visuales además de facilitar la tarea de reconocer y recordar los diagramas y mejorar la legibilidad de los mismos, especialmente para usuarios no expertos. Además, la inclusión de algunos objetos dentro de otros (a través de límites entre elementos) crea relaciones de agrupación cuando se perciben los elementos, lo que permite expandir el número de elementos que pueden ser mostrados en cada diagrama sin exceder los límites cognitivos. Este mecanismo es complementado mediante el uso de los paquetes proporcionados por UML. Estudios como el de Moody [183] indican que estos mecanismos de agrupación contribuyen a mejorar la precisión a la hora de entender y verificar los diagramas en hasta un 50 %.
- **Precedencia perceptual**: en la notación propuesta, no todos los conceptos en el metamodelo MRM tienen asociado un icono. Dado el hecho de que la memoria de trabajo humana es limitada, ésta necesita establecer el orden en el que se trata con los elementos. La definición de algunos conceptos por medio de etiquetas en lugar de con nuevos estereotipos ayuda a elegir a qué elementos prestar atención en primer lugar. Por ejemplo, si observamos el estereotipo *Medida*, éste mapea dos metaclases de MRM (*Medida Base* y *Medida Derivada*). Los elementos representados son discriminados a través de valores etiquetados asociados con el estereotipo.
- **Familiaridad perceptual (conocimiento de la notación):** el hecho de que la notación propuesta utilice los iconos de GRL (que, a su vez, son compartidos por la mayoría de propuestas basadas en *i**) y que incluso reutilice conceptos de GRL (y no sólo notación) cuando ha sido posible (por ejemplo, con el elemento *meta*), facilita su reconocimiento por parte de analistas familiarizados con estas notaciones.

Los lectores interesados pueden encontrar información adicional sobre este perfil así como del soporte automático diseñado para el mismo en [25], adjuntado como apéndice en este documento. Para ilustrar el uso del mismo, la siguiente sección muestra su uso a través de un ejemplo.

Elemento GRL	Notación	Equivalencia en MRM
Actor		Participante
Meta *(goal)*		Meta, Indicador, Medida Base, Medida Derivada
Meta Difusa *(softgoal)*		Requisito No Funcional
Tarea *(task)*		Requisito Funcional , Modelo de Análisis, Función de Medición, Método de Medición
Recurso *(resource)*		Atributo, Clase de Entidad, Criterio de Decisión
Dependencia de Tarea *(Task Dependency)*		Caso de Test
Dependencia de Recurso *(Resource Dependency)*		Escenario de Calidad, Resultado de Medición

Tabla 4.3: Representando las metaclases de MRM por medio de GRL

4.6. Ejemplo de uso

El ejemplo seleccionado para ilustrar nuestra propuesta corresponde a una versión simplificada de un sistema *online* de venta de entradas de cine. A través de este ejemplo se mostrará tanto las instanciación de los metamodelos propuestos como la notación asociada.

4.6.1. Descripción

Supongamos que el *gerente de ventas* de una cadena de cines desea *incrementar los beneficios de las ventas* de la compañía. Supongamos también que esta meta puede ser conseguida a través de dos submetas diferentes. Por un lado, la compañía quiere *incrementar el beneficio neto de ventas* mediante la disminución del coste asociado a la venta de cada entrada. Por otro lado, la compañía también desea *incrementar el número de ventas*, lo que implica incrementar su número de clientes potenciales. El gerente de ventas cree que ofrecer la posibilidad de adquirir entradas mediante un sistema electrónico de venta a través de Internet puede influir de forma positiva en ambas submetas, así que ha decidido abordar el proyecto de construcción de dicha aplicación.

Los clientes web que interactuarán con la aplicación tendrán como principal requisito funcional el de *gestionar la compra de entradas*. Este requisito puede ser descompuesto en dos requisitos funcionales: *visualizar cartelera* y *comprar entradas*. Además, se han identificado varios requisitos no funcionales: la funcionalidad de gestionar compra de entradas debe seguir las guías de *accesibilidad* de la W3C (*World Wide Web Consortium*

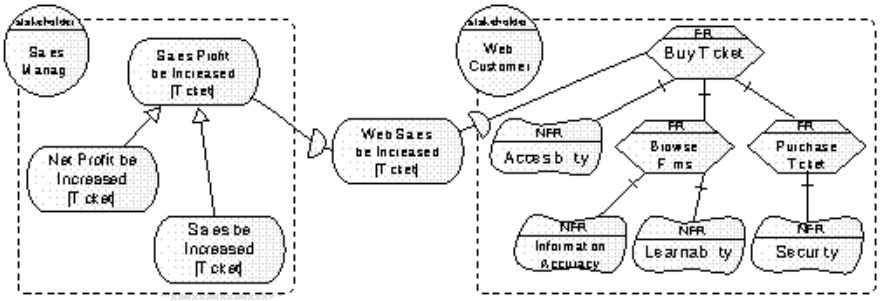

Figura 4.8: Requisitos para el modelo de venta *online* de entradas de cine

[187]) para permitir que un mayor rango de usuarios puedan acceder al sistema. Además, la aplicación debe ser *fácil de aprender*, es decir, debe ser lo suficientemente simple como para que usuarios noveles puedan comprar entradas desde su primera interacción con el sistema. De forma adicional, el sistema deberá proporcionar *información precisa* mientras se visualiza la cartelera: los datos correspondientes a sinopsis, sesiones, precios, etc. deben ser fidedignos. Por último, la funcionalidad de compra de entradas debe ser realizada de forma que se garantice la *seguridad* y confidencialidad de los datos de los clientes.

4.6.2. Instanciación de requisitos mediante el perfil para MRM

Si analizamos las metaclases de URM implicadas en esta descripción (ver Figura 4.1), podemos observar como (i) el gerente de ventas y el comprador web son instancias de la metaclase `Participante`, (ii) todas las metas relacionadas con la compra de entradas instancian la metaclase `Meta`, (iii) gestionar compra, visualizar cartelera y comprar entradas son `Requisitos Funcionales` y (iv) accesibilidad, facilidad de aprendizaje, precisión de información y seguridad son todas instancias de `Requisito No Funcional`.

Toda esta información puede ser modelada con ayuda del perfil presentado anteriormente y, más concretamente, con la parte de este perfil correspondiente al metamodelo de requisitos. El modelo correspondiente a la descripción anterior es mostrado en la Figura 4.8.

4.6.3. Instanciación de medidas mediante el perfil para MRM

Centrémonos ahora en el requisito no funcional que aborda la facilidad con la que un comprador web es capaz de comprar entradas en el sistema y veamos cómo este requisito se puede convertir en un requisito medible. Para ello, asumamos que hemos definido un escenario de calidad (instanciando la metaclase `Escenario de Calidad`, ver Figura 4.1) que establece que es necesario asegurar que un usuario novel encuentra, según navega por el sistema, los enlaces que espera en base a su modelo mental del dominio.

Analizando el metamodelo SMM– (Figura 4.3) observamos como el primer paso para hacer un requisito medible es definir una ocurrencia de `Indicador`, que en este caso hemos denominado *nivel de navegabilidad* (Figura 4.10). Los indicadores están compuestos de medidas. En nuestro ejemplo, supongamos que hay una única instancia de `Medida Derivada` que contribuye al indicador, llamada *Medida de Cobertura Navegacional de Dominio* (DCNM en la Figura 4.10). Esta medida está basada en dos ocurrencias de `Medida Base`: el *número de relaciones de dominio relevantes* (NRR) y el *número de relaciones de dominio navegables* (NNDR). Brevemente, la DCNM asume que el modelo mental de dominio del usuario coincide con el modelo de dominio de la aplicación y, a partir de esa asunción, comprueba cuántas de las relaciones de dominio relevantes para cumplir el requisito están reflejadas en el modelo de navegación de la aplicación mediante enlaces por los que el usuario puede navegar. El razonamiento lógico completo que hay detrás de esta medida se puede consultar [175].

Supongamos también que, en este contexto, el `Criterio de Decisión` y el `Modelo de Análisis` asociados con el indicador establecen que una DCNM > 0.80 hace que el diseño de la interfaz web de la aplicación sea aceptable con respecto a su facilidad de aprendizaje. A la hora de calcular la medida, la `Forma de Medir` dependerá de la `Categoría de Entidad` sobre la que deseemos realizar dicha medición. Puesto que una detección temprana de incumplimiento de criterios de calidad reduce los costes asociados a su corrección, asumamos que disponemos de los modelos web conceptuales de dominio y de navegación correspondientes a esta aplicación. Este tipo de modelos son proporcionados por cualquier aproximación de Ingeniería Web y se corresponden con las ocurrencias del concepto `Categoría de Entidad`. Además, ambas medidas están relacionadas con el `Atributo` llamado *Complejidad Estructural* de los modelos web conceptuales. La `Función de Cálculo` que sirve para calcular la DCNM utiliza las dos `Medidas Base` (NRR y NNDR). Cuanto mayor sea valor de DCNM, más relaciones de dominio estarán representadas mediante enlaces en la interfaz de la aplicación web. Obviamente, la prueba final de que la aplicación cumple con los requisitos del cliente se obtiene mediante la definición de un `Caso de Prueba` que demuestre que un comprador novel consigue comprar la entrada (otro `Indicador`, esta vez de tipo *booleano* y medible sobre la aplicación en ejecución, y cuya descripción completa se ha omitido en la Figura 4.10).

La Figura 4.9 muestra el diagrama de objetos correspondiente a la instanciación de MRM para el ejemplo descrito. Como se observa en la figura, a pesar de que la complejidad del ejemplo no es muy elevada, el tamaño de los diagramas de objetos obtenidos es grande, mientras que su nivel de comprensibilidad es bajo. Por ello, dotar a la propuesta de una notación visual permite la construcción de modelos más concisos al mismo tiempo que se mejora la facilidad de comprensión de los mismos. Además, el alineamiento y adaptación de una propuesta basada en *i** hace que los modelos de requisitos medibles presenten una apariencia bastante similar a los que se pueden construir con esta notación simplificándose así tanto la comprensión como las construcción de nuevos para modeladores familiarizados con notaciones basadas en *i**.

Estos modelos podrían ser usados en el contexto de las metodologías de desarrollo dirigidas por modelos para automatizar el proceso de medición en cada etapa del proceso de desarrollo e incluso para hacer evolucionar automáticamente los modelos basándose

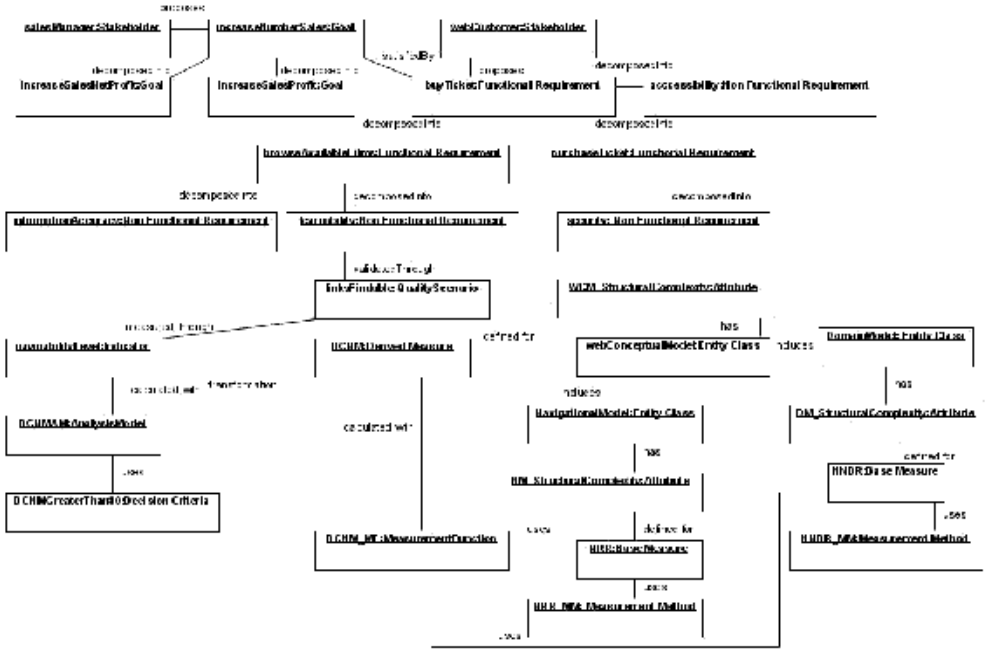

Figura 4.9: Modelo de objetos para el sistema de venta *online*

en los resultados obtenidos en la medición [175].

4.7. URM y requisitos de seguridad

4.7.1. Introducción

Como se comentó en la Sección 2.3.1, la seguridad se ha convertido en un factor crítico tanto para el éxito de muchos proyectos de desarrollo software como para la supervivencia de multitud de organizaciones. A pesar de ello, los requisitos de seguridad todavía no suelen recibir la atención que merecen, siendo éste un aspecto que, en muchas ocasiones, se trata una vez que el sistema ya ha sido desarrollado en lugar de considerarlos durante todas las fases de su ciclo de desarrollo [77, 78, 79]. De forma reciente ha emergido un área de trabajo denominada *seguridad dirigida por modelos* con el objetivo de aplicar las técnicas introducidas por los paradigmas *model driven* en el modelado e implementación de aspectos de seguridad de sistemas software.

A lo largo de este capítulo, y también de forma alineada con las propuestas *model driven*, se ha incidido en la consideración temprana de los requisitos del sistema a través de técnicas como el metamodelado de requisitos. Dado que, por un lado, se disponía del metamodelo URM orientado a la elicitación de requisitos (ver Sección 4.4) y que, por otro lado, la consideración de requisitos de seguridad debe tratarse desde la fase de elicitación de requisitos del sistema, como otro objetivo de esta tesis se trabajó en la extensión de URM con conceptos específicos que permitieran la definición temprana

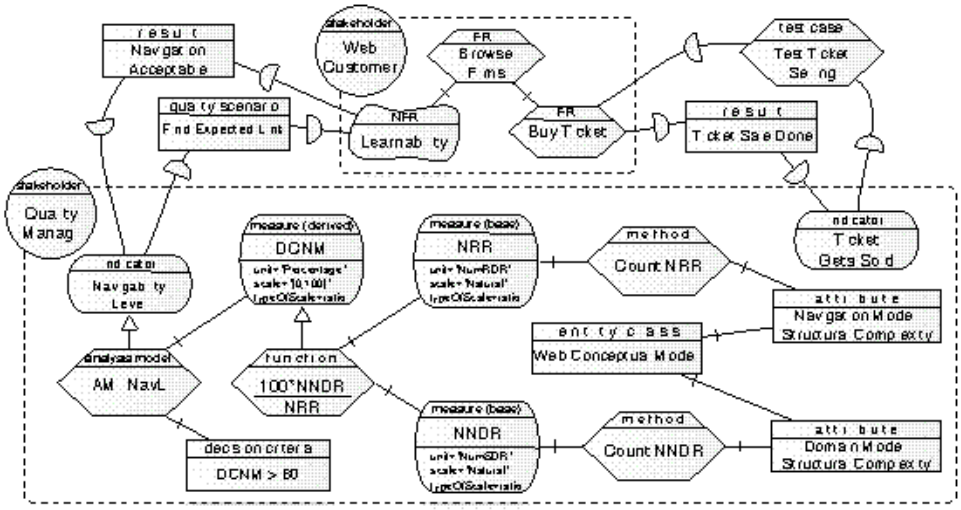

Figura 4.10: Modelado mediante el perfil para el sistema de venta *online*

de requisitos de seguridad. Estos conceptos se explican en la Sección 4.7.2 y han sido extraídos de estándares como el ISO/IEC 15408 (también conocido como *Common Criteria Framework* [188]) y de propuestas surgidas en este ámbito como [189, 190, 191]. Como resultado de esta extensión se ha obtenido un metamodelo de requisitos extendido a partir del que se ha diseñado un lenguaje específico de dominio (DSL) que permite la gestión de requisitos de seguridad. Este DSL estará en la base de una arquitectura denominada ModelSec que se alinea con el paradigma MDS para permitir la generación de artefactos software relacionados con seguridad (como, por ejemplo, código que implementa reglas de seguridad) a través de una cadena de transformaciones que utiliza como entrada los requisitos elicitados.

4.7.2. Añadiendo conceptos de seguridad a URM

En la definición de URM (ver Figura 4.1) toman parte una serie de conceptos que facilitan la identificación de requisitos y metas para el sistema que pueden describirse mediante el uso de términos bien definidos. El propósito que se aborda ahora es extender dicho metamodelo con el objetivo de contribuir a la elicitación temprana de requisitos relacionados con la seguridad del sistema. Para conseguir este objetivo se ha propuesto una extensión de URM realizada mediante un nuevo metamodelo definido a partir de conceptos propios del ámbito de la seguridad como es el caso de los recursos a ser protegidos, las amenazas que podrían aparecer sobre dichos recursos o los mecanismos de control de acceso a los mismos. Esta extensión de URM permitirá una descripción detallada de aquellos requisitos relacionados con la seguridad. Además, estos requisitos podrán ser transformados posteriormente a otros artefactos software como, por ejemplo, código que pueda tomar parte del sistema para asegurar el cumplimiento de los requisitos de seguridad elicitados.

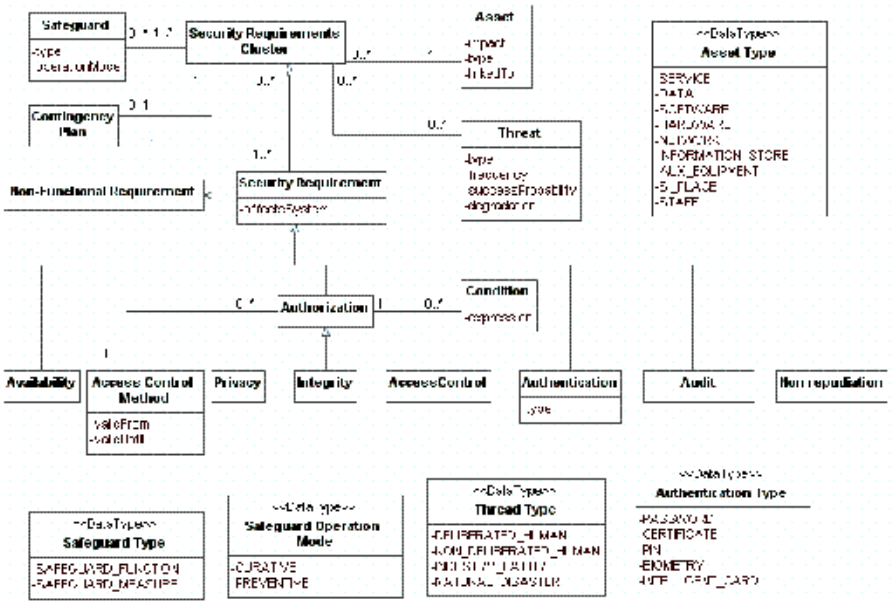

Figura 4.11: Extendiendo URM con conceptos generales de seguridad

Describamos ahora los conceptos que han sido añadidos a URM. Para simplificar la explicación de los mismos, estos conceptos se han clasificado en tres categorías (conceptos básicos de seguridad, requisitos de seguridad y métodos de control de acceso) y se muestran en dos figuras separadas (Figuras 4.11 y 4.12), estando la primera de ellas centrada en los conceptos generales de seguridad y la segunda en los métodos de control de acceso.

En primer lugar se describirán los elementos que aparecen en la Figura 4.11. Conforme a las especificaciones del estándar ISO/IEC 15408, los conceptos básicos de seguridad considerados han sido `Asset` (recurso), `Threat` (amenaza), `Safeguard` (salvaguarda) y `Contingency Plan` (plan de contingencia). Un recurso es un objeto físico o lógico que tiene un valor para la organización. Estos recursos pueden ser de diferentes tipos (por ejemplo, documentos, tablas de datos, etc.) y suelen tener asociado un índice de impacto que ayuda a medir la importancia que cada recurso tiene dentro de la organización. Los recursos pueden verse dañados por diferentes amenazas que tienen propiedades como su tipo, la probabilidad y frecuencia de ocurrencia (modelada esta última como una tasa anual), o el nivel de degradación (entendido como el grado de daño causado sobre un recurso si la amenaza llega a materializarse).

Las salvaguardas tienen el propósito de reducir riesgos sobre recursos. Como se muestra en la Figura 4.11, es posible distinguir diferentes tipos de salvaguardas denominadas `Safeguard Functions` (acciones para reducir un riesgo) y `Safeguard Measures` (dispositivos o procesos lógicos para reducir un riesgo). Las salvaguardas también suelen clasificarse en preventivas (si actúan antes de que se materialice una amenaza) o curativas (si se aplican sobre recursos que ya han sido dañados). Las salvaguardas suelen agruparse formando planes de contingencia con el propósito de reducir los daños que una amenaza puede causar sobre un recurso.

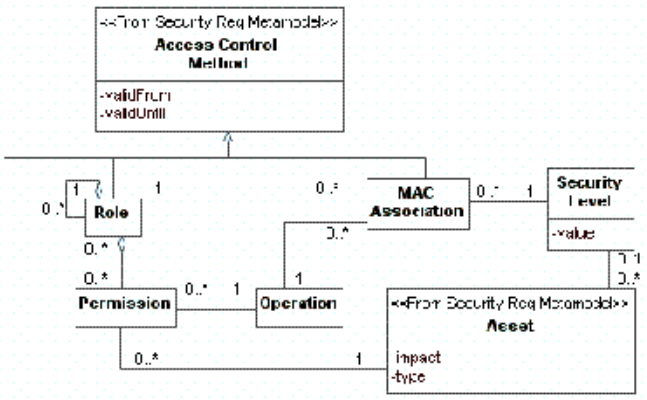

Figura 4.12: Extendiendo URM con mecanismos de control de acceso

En lo que respecta a los tipos de requisitos de seguridad, hay que indicar que no existe una clasificación estándar para ellos, aunque ciertas categorías son habitualmente consideradas en este ámbito. En nuestro caso, y siguiendo la clasificación propuesta por Rodríguez *et al.* [189], se han considerado siete categorías de requisitos de seguridad que son privacidad, integridad, control de acceso, autenticación, disponibilidad, no repudio y auditoría. La privacidad se centra en asegurar que la información sólo puede ser leída por aquellos usuarios que están autorizados para ello. Los requisitos de integridad tienen el propósito de garantizar que la información se mantiene completa y correcta. Los mecanismos de control de acceso son usados para establecer restricciones acerca de qué usuarios pueden acceder a cada recurso.

La autenticación se centra en las partes implicadas en una comunicación o interacción, pudiendo clasificarse en dos tipos: autenticación de los usuarios, también conocida como autenticación del destino, y autenticación de la fuente de datos, también conocida como autenticación de la fuente. Los requisitos de disponibilidad se centran en asegurar que los usuarios autorizados pueden acceder a la información a la que tienen acceso cuando así lo necesiten mientras que los de no repudio controlan a las partes implicadas en una comunicación o acción de forma que éstas no puedan negar su participación en la misma. Por último, la auditoría intenta registrar el uso que se hace de los recursos del sistema. Todos estos tipos de requisitos pueden afectar tanto a todo el sistema como a recursos concretos del mismo.

Habitualmente, los requisitos pueden ser agrupados debido a que están relacionados con un mismo recurso, reducen los efectos de una misma amenaza o persiguen un mismo objetivo de seguridad. Este concepto de agrupación, usado en propuestas como la de Mellado *et al.* [191], ha sido introducido en nuestro metamodelo a través del concepto de `Security Requirements Cluster` (ver Figura 4.11).

Pasemos ahora a describir los conceptos del metamodelo mostrados en la Figura 4.12. Con respecto a los requisitos relacionados con privacidad, integridad y control de acceso, han sido directamente asociados con un mecanismo de control de acceso (`Access Control Method` en la Figura 4.12). Los diferentes métodos considerados han sido `permisos`

(*Discretionary Access Control*, DAC), `niveles de seguridad` (*Mandatory Access Control*, MAC) o `roles` (*Hierarchical Role-Based Access Control*, HRBAC) [190]. El concepto de `MAC Association` ha sido introducido para asociar un nivel de seguridad y una operación con un rol.

Todos los conceptos mencionados han sido agrupados en un metamodelo (Figuras 4.11 y 4.12) que ha servido para extender URM con conceptos específicos del ámbito de la seguridad. El punto de extensión para añadir estos conceptos a URM es la metaclase `Requisito No Funcional`, siendo la metaclase `Security Requirement` una especialización de `Requisito No Funcional` que sirve como raíz para el resto de metaclases que representan conceptos de seguridad.

4.7.3. ModelSec: una propuesta para la gestión de requisitos de seguridad

Como mencionamos anteriormente, el metamodelo descrito en la sección anterior ha sido usado como base para la creación de un DSL que permite la construcción de modelos de requisitos de seguridad y que forma parte de una arquitectura generativa, denominada ModelSec, orientada a la gestión de requisitos de seguridad. A continuación, ofreceremos una descripción general de esta arquitectura, centrándonos principalmente en los aspectos de definición de requisitos de seguridad y uso de metamodelos de requisitos por ser éstas las áreas más relacionados con esta tesis doctoral. Todos los detalles acerca de esta arquitectura y un ejemplo completo de uso pueden encontrarse en [30], que se encuentra al final de este documento.

Mientras que, de forma mayoritaria, las propuestas surgidas en el ámbito de MDS (ver Sección 2.3.1) se basan en el uso de perfiles, en este caso se propone el uso de un DSL gráfico, una práctica más habitual en los últimos años debido a que ofrece mayor flexibilidad que el uso de perfiles [90]. Un DSL consiste básicamente en tres elementos: un metamodelo de sintaxis abstracta, una sintaxis concreta y una semántica. La sintaxis abstracta define los conceptos que toman parte en el DSL, las relaciones entre ellos y las reglas que indican cómo los modelos deben ser creados. La sintaxis concreta define una notación para la sintaxis abstracta mientras que la semántica se proporciona, generalmente, por medio de transformaciones que permiten trasladar un modelo expresado mediante el uso del DSL a modelos expresados en lenguajes con una semántica bien definida [89].

Para la implementación de nuestro DSL de gestión de requisitos de seguridad se ha hecho uso de la plataforma Eclipse. Como se describió en la Sección 2.6, Eclipse ofrece mecanismos adecuados para la definición de metamodelos y editores gráficos a través de EMF y GMF, además de una integración sencilla con lenguajes de soporte a transformaciones modelo a modelo y modelo a código.

En la base de este DSL se encuentra URM y su extensión para la definición de requisitos de seguridad descrita en la Sección 4.7.2. Este metamodelo ha sido definido en EMF mientras que los editores gráficos para construir modelos conformes con este metamodelo han sido implementados usando GMF. Para facilitar su uso por parte de los usuarios, el

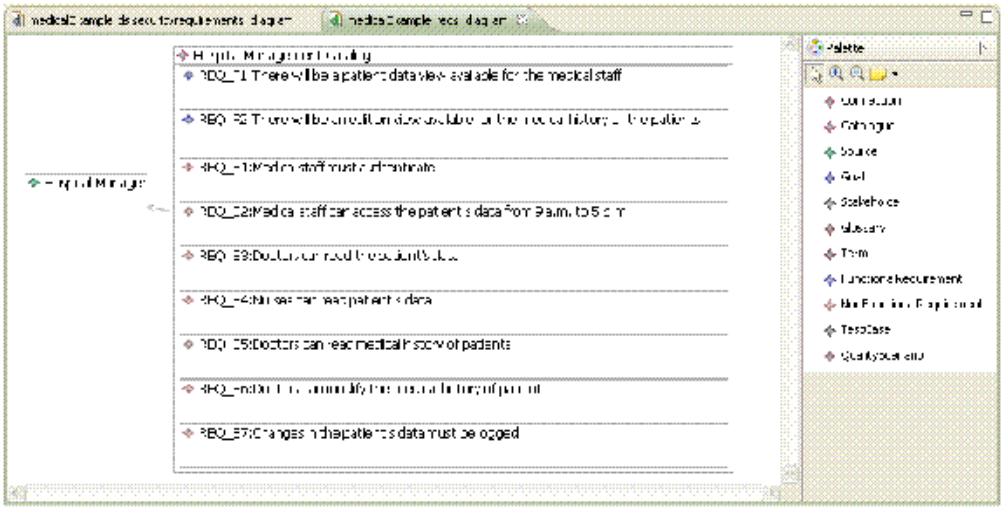

Figura 4.13: Un ejemplo de definición de requisitos de seguridad

DSL ha sido dividido en dos vistas conectadas: mientras que una de las vistas se centra en los conceptos generales de requisitos (es decir, aquellos que recogía URM), la otra está enfocada a los conceptos de seguridad descritos en la Sección 4.7.2. En la Figura 4.13 se muestra la apariencia de este DSL mediante un ejemplo. Concretamente, este caso corresponde a la identificación de requisitos para una aplicación web que gestiona información médica de pacientes y que fue adaptado de Fernández-Medina *et al.* [192]. Como puede observarse en la Figura 4.13, el DSL ha sido utilizado para describir requisitos correspondientes a las capacidades de acceso a la información clínica de los pacientes, restringiéndose qué miembros de la plantilla del hospital pueden acceder a ellos, en qué horario y con qué capacidades.

Una vez identificados los requisitos de seguridad del sistema, se puede trabajar en la descripción más detallada de los mismos haciendo uso de la parte del metamodelo correspondiente a conceptos de seguridad. De esta forma, será posible especificar, por ejemplo, los recursos que deben ser protegidos, las amenazas sobre ellos o los mecanismos de control de acceso a los mismos. Un extracto del modelo que describe estos elementos se puede observar en la Figura 4.14.

En la Figura 4.14 se ha establecido que los principales recursos que deben ser protegidos son las tablas `Patient` y `Medical History`, en las que se almacena información personal y clínica de los pacientes y que está sujeta, por tanto, a requisitos específicos de seguridad. Las principales amenazas sobre las que hay que proteger a estos recursos consisten en el acceso a los mismos por parte de personal no autorizado. Para evitarlo, se ha dividido el personal clínico en diferentes *roles* (doctores y enfermeros/as). Cada *rol* tendrá asociado unos permisos mientras que se ha asociado a cada recurso un nivel de seguridad que, en este caso, ha sido definido como *confidencial* para la información personal de los pacientes y *sensible* en el caso de su información médica (por ejemplo, diagnósticos o tratamientos). Sólo aquellos roles con un nivel de autorización suficiente podrán acceder a cada uno de los recursos. Como puede observarse en la Figura 4.14, también se ha hecho

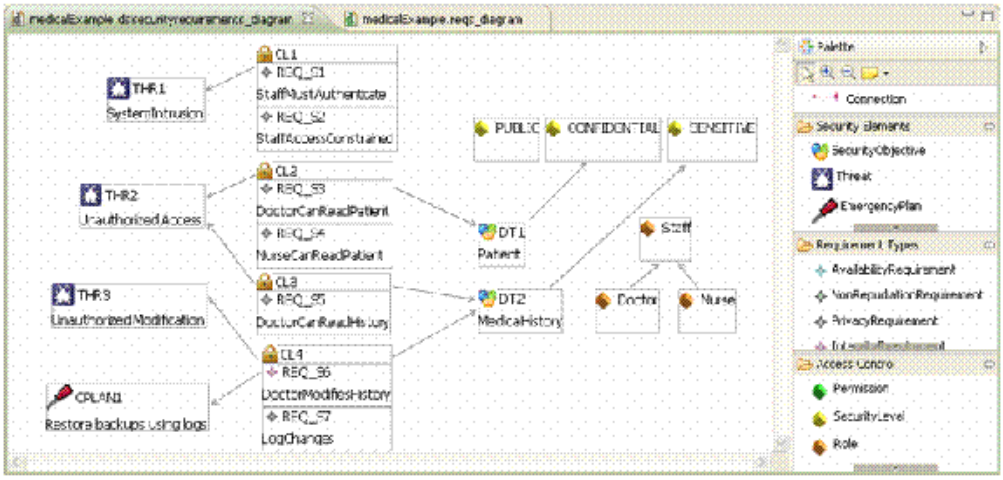

Figura 4.14: Extracto de un modelo de definición de requisitos de seguridad

uso del concepto de *cluster* (*CL*) para agrupar requisitos de seguridad relacionados entre sí. Una descripción más detallada de este ejemplo puede ser encontrada en la Sección 6 de [30], adjunto en el Apéndice *'Publicaciones'*.

Uno de los mayores potenciales de ModelSec reside en el hecho de que los modelos de requisitos creados con el DSL como, por ejemplo, el modelo de la Figura 4.14, servirán como punto de partida para una arquitectura generativa que, en base a transformaciones entre modelos, permite llegar desde los requisitos identificados hasta código orientado al cumplimiento de los mismos. Para ello, ha sido necesario definir un conjunto de modelos adicionales que permiten ir introduciendo decisiones de diseño y detalles específicos de las plataformas para las que se desea generar el código correspondiente a estos requisitos. De esta forma, y alineándose con el paradigma MDS, la arquitectura parte de modelos con un alto nivel de abstracción que no tienen detalles específicos de ninguna plataforma y que van transformándose de forma progresiva a modelos de nivel de abstracción más bajo que incluyen decisiones de diseño y detalles específicos de las plataformas para las que se genera código.

En la Figura 4.15 se muestra un visión general de esta arquitectura. Como se aprecia en la parte superior de la misma, una vez que los modelos de requisitos de seguridad han sido creados, a partir de ellos es posible crear de forma automática esqueletos de modelos de diseño y modelos de implementación sobre los que los modeladores pueden añadir este tipo de decisiones. Una vez que se hayan creado estos artefactos y se hayan completado con decisiones de diseño e implementación, los modelos pasan a otra cadena de transformaciones definida verticalmente en la Figura 4.15. El propósito de la misma es generar artefactos software a partir de los modelos de seguridad creados previamente. Para ello, los modelos de requisitos de seguridad, diseño e implementación sirven como entrada para transformaciones modelo a modelo (M2M) que generan varios modelos de la plataforma destino. El lenguaje RubyTL [193] ha sido utilizado para la implementación de estas transformaciones. Por último, se alcanza una etapa de transformaciones modelo a código (M2C) implementadas en el lenguaje MofScript [154] y que permiten obtener

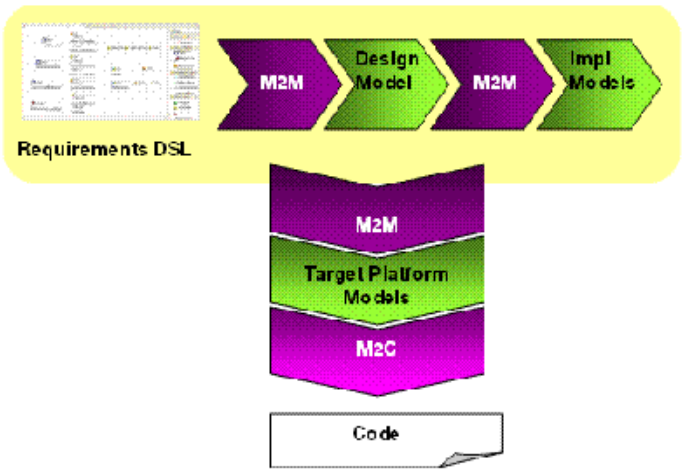

Figura 4.15: Visión general de la arquitectura generativa

código fuente relacionado con los requisitos de seguridad elicitados inicialmente y que puede corresponder, por ejemplo, con políticas de seguridad y acceso a la base de datos del sistema o políticas de seguridad para aplicaciones web.

En la Figura 4.16 se muestra un fragmento de código generado automáticamente para el ejemplo mostrado en Figura 4.14. Concretamente, el código generado está orientado a reflejar sobre un sistema de bases de datos Oracle [194] los requisitos elicitados con respecto al acceso a la información almacenada en la tabla de pacientes, sobre la que se establecen diferentes restricciones de acceso.

Una información más detallada sobre esta arquitectura puede encontrarse en [30]. En este capítulo nos hemos centrado principalmente en los aspectos relativos a la identificación de requisitos de seguridad mediante técnicas de metamodelado y a su uso como base de la arquitectura de ModelSec. Esta línea de trabajo comenzó a abordarse en los últimos meses de realización de esta tesis doctoral junto con otro doctorando. Por el momento, se ha conseguido generar código para plataformas como Oracle y, en el futuro, se debe trabajar con el propósito de ofrecer soporte tanto para otras plataformas como para la generación de otros mecanismos de control de acceso y políticas de seguridad. Desde nuestro punto de vista, este trabajo tiene entidad suficiente para servir de área de estudio para, al menos, otra tesis doctoral.

```
execute sa_sysdba.create_policy('Access_Patient', 'ols_column', 'read_control,label_default,hide');
execute sa_components.create_level('Access_Patient', 1000, 'PUB', 'PUBLIC');
execute sa_components.create_level('Access_Patient', 2000, 'CONF', 'CONFIDENTIAL');
execute sa_components.create_level('Access_Patient', 3000, 'SENS', 'SENSITIVE');
execute sa_label_admin.create_label('Access_Patient', 1000, 'PUB');
execute sa_label_admin.create_label('Access_Patient', 2000, 'CONF');
execute sa_label_admin.create_label('Access_Patient', 3000, 'SENS');
execute sa_user_admin.set_user_labels('Access_Patient', 'PClarkson', 'CONF', 'CONF', 'PUB', 'CONF', 'CONF');
execute sa_user_admin.set_user_labels('Access_Patient', 'SWilliams', 'CONF', 'CONF', 'PUB', 'CONF', 'CONF');
execute sa_user_admin.set_user_labels('Access_Patient', 'BMorcon', 'CONF', 'CONF', 'PUB', 'CONF', 'CONF');
execute sa_policy_admin.apply_table_policy('Access_Patient', 'test', 'Patients');
```

Figura 4.16: Fragmento del código generado automáticamente para Oracle

Capítulo 5

Evaluación temprana de usabilidad

5.1. Introducción

Tras haber descrito en el Capítulo 4 las contribuciones de esta tesis a la fase de gestión de requisitos, los siguientes capítulos se centrarán en la fase modelado. En este capítulo se comenzará mostrando una propuesta diseñada para dar soporte a la definición y evaluación de requisitos y métricas relacionadas con la usabilidad del software en tiempo de modelado y se mostrará su aplicación utilizando los modelos conceptuales usados por las metodologías de desarrollo de sistemas web.

5.2. Usabilidad sobre modelos navegacionales

Como se mencionó en el Capítulo 2, en la literatura existen múltiples definiciones de usabilidad, hecho que ha motivado que este atributo del software pueda ser interpretado desde diferentes perspectivas y que, incluso, diferentes *stakeholders* de un mismo sistema la perciban de formas diferentes [131]. Además, estas múltiples definiciones también muestran que la usabilidad es un atributo de calidad abstracto que depende de muchos factores. Por ello, si se desea mejorar la usabilidad de un sistema software se puede incidir en múltiples aspectos de éste.

En esta tesis, la usabilidad será tratada desde la perspectiva de un usuario final, considerándola como un atributo que ayuda a valorar si los usuarios pueden llevar a cabo las tareas para las que se concibió el sistema de una forma intuitiva y eficiente. Aunque esta consideración es ligeramente más precisa que las definiciones generales de usabilidad mostradas en el Capítulo 2, ésta continúa afectando a numerosas características como la navegación en el sistema, la facilidad de uso, la simplicidad para llevar a cabo tareas o la presentación atractiva y cómoda para los usuarios. Concretamente, en el marco de esta tesis se propondrá una estrategia orientada a la mejora de la usabilidad de un sistema a través de la mejora en la navegación y el acceso a la información y funcionalidades que éste presenta, es decir, ofreciendo una navegación intuitiva que simplifique la forma en la que los usuarios para los que el sistema fue concebido llevan a cabo sus tareas.

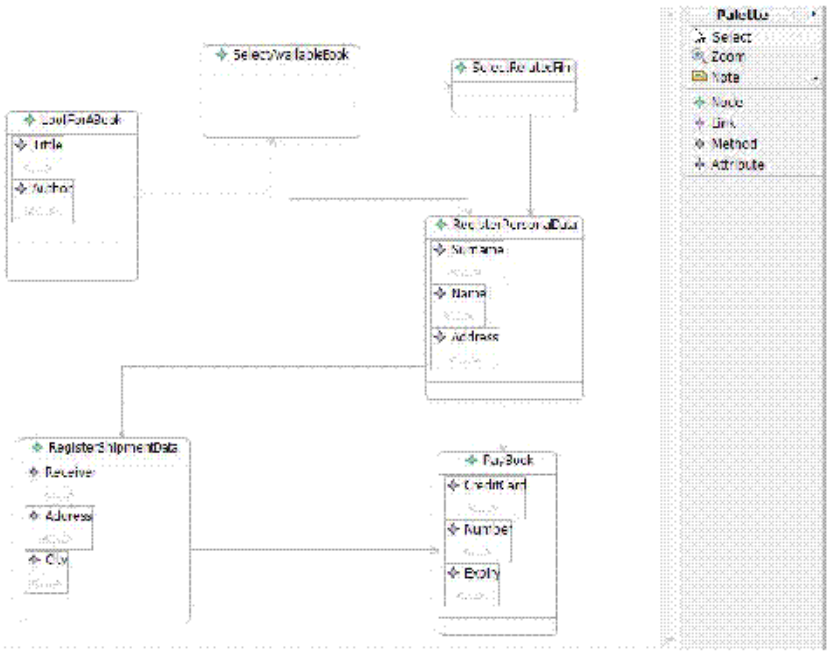

Figura 5.1: Fragmento de un modelo navegacional para un sistema *online* de venta de libros

Para el desarrollo de sistemas web, las metodologías existentes proponen la utilización de diferentes modelos como modelos de comportamiento, modelos de presentación o modelos navegacionales, que son los usados para modelar la navegación de los usuarios en el sistema y en los que se centra la propuesta presentada. En todas estas metodologías se considera necesario el tratamiento de la navegación, resultando éste un aspecto crítico para el éxito de los sistemas desarrollados [195]. Aunque como resalta Escalona *et. al* [195], los modelos navegacionales propuestos por cada metodología presentan diferencias, generalmente están compuestos por dos elementos principales: nodos y enlaces (*links*) entre nodos. Cada nodo es usado para representar una información o funcionalidad que será presentada a los usuarios del sistema mientras que los enlaces son utilizados para unir nodos, indicando así la posibilidad de navegar desde un nodo que representa una información o funcionalidad hacia otro. Por tanto, un modelo navegacional es similar a un grafo dirigido, donde los nodos representan información o funcionalidad mientras que las aristas representan las rutas que un usuario puede seguir durante su navegación a través del sistema.

Como ejemplo, en la Figura 5.1 se muestra un sencillo fragmento de un modelo navegacional para un sistema dedicado a la compra de libros por Internet que presenta la información y funcionalidades relacionadas con este proceso de compra como puede ser buscar un libro, seleccionar un libro entre los disponibles para su compra o realizar el pago del mismo. Además, en estos modelos podrían aparecer otras estructuras como menús o índices pero no se muestran en la Figura 5.1 para facilitar la comprensión de la propuesta.

La calidad de los modelos navegacionales resulta importante ya que éstos representan los posibles caminos que los usuarios podrían seguir durante su navegación por el sistema. Por tanto, errores en estos modelos o diseños navegacionales incorrectos influyen en la usabilidad del sistema final. En la literatura pueden encontrarse diferentes trabajos centrados en la evaluación y mejora de la calidad de los modelos navegacionales. Por ejemplo, en Abrahao *et al.* [196] se define un conjunto de métricas que permiten a los modeladores obtener información acerca de la calidad de sus modelos navegacionales. Con un objetivo similar, y también como parte de esta tesis doctoral, en [31, 32] se propone una estrategia para la verificación y demostración de propiedades sobre modelos navegacionales, que será explicada en detalle en el Capítulo 6.

Según demuestran estudios empíricos como el realizado por Escalona *et al.* [195], los requisitos relacionados con la navegación del sistema deben ser considerados desde fases iniciales del desarrollo del mismo. Sin embargo, en ciertas ocasiones los *stakeholders* establecen requisitos que influyen en la navegación del sistema pero que no pueden ser expresados sobre los modelos navegacionales propuestos, lo que hace que muchas veces sean olvidados o no sean evaluados hasta que el sistema está completamente desarrollado. En la siguiente sección se analizarán en más detalle estos requisitos y la propuesta abordada para permitir su definición y evaluación en tiempo de modelado.

5.3. Expresando requisitos de usabilidad sobre modelos navegacionales

Durante la elicitación de requisitos para sistemas de información web, los *stakeholders* implicados proponen ciertos requisitos relacionados con la usabilidad del sistema que, sin embargo, no pueden ser expresados sobre los modelos conceptuales diseñados para el modelado del sistema. Un estudio realizado por Atterer y Schmidt [137] resaltó la existencia de este tipo de requisitos y, además, la aplicación de las técnicas para la elicitación de requisitos presentadas en el Capítulo 4 nos permitieron descubrir nuevos requisitos de este tipo.

Algunos de estos requisitos están relacionados con el acceso a la información y funcionalidades presentadas en el sistema. Por ejemplo, en ciertas ocasiones los *stakeholders* desean establecer el número máximo de *clicks* que un usuario necesitará para llevar a cabo una determinada tarea en el sistema. En otras ocasiones desean expresar que no toda la información y funcionalidades presentadas tienen la misma importancia ya que, generalmente, en un sistema de información web conviven datos y funcionalidades relacionadas con el objetivo principal para el que está diseñado y otras que pueden ser consideradas como secundarias. Por ejemplo, en una aplicación web para la compra *online* de billetes de avión, las funcionalidades primarias son aquellas relacionadas con la búsqueda de vuelos y compra de billetes mientras que pueden existir otras consideradas secundarias como realizar una reserva de hotel o un alquiler de coche en el destino del vuelo. Estas últimas funcionalidades son importantes para el sistema pero no representan el objetivo principal del mismo así que los *stakeholders* suelen desear que las informaciones y funcionalidades primarias estén más cercanas y accesibles a los usuarios, por

ejemplo, en términos de distancia desde el punto de entrada al sistema, facilitando así a los usuarios la navegación y mejorándose la usabilidad del software construido.

Otros requisitos que los *stakeholders* suelen proponer establecen restricciones en el orden en el que los usuarios visitan las funcionalidades del sistema o también pueden requerir que se obligue a la existencia de conectividad directa o indirecta entre nodos que representan información o funcionalidades relacionadas, bajo la premisa de que estos requisitos facilitarán la navegación de los usuarios por el sistema y, por tanto, mejorarán su usabilidad. En la Tabla 5.1 se muestran algunos de estos requisitos, que fueron presentados en [22, 28].

Si los requisitos del tipo mostrado en la Tabla 5.1 no son expresados durante el modelado del sistema se corre el riesgo de que sean olvidados y no se tengan en cuenta durante el proceso de desarrollo. Este hecho puede implicar que el sistema final no ofrezca una facilidad de uso y navegación tan intuitiva y eficiente como podría, lo que puede motivar que sus usuarios se sientan perdidos y desorientados durante su uso. Además, también se pierde la posibilidad de aprovechar las ventajas ofrecidas por la evaluación de usabilidad en tiempo de modelado, como la solución efectiva en tiempo y coste de problemas más complicados de resolver en fases posteriores del ciclo de vida.

Ante esta situación, en esta tesis se propone una extensión de los mencionados modelos navegacionales que permita la definición de este tipo de requisitos y su evaluación en tiempo de modelado. Esta propuesta, que también se presentó en [22], se detalla en las siguientes secciones y se ilustra utilizando los requisitos mostrados en la Tabla 5.1.

5.3.1. Extendiendo los metamodelos navegacionales con características de usabilidad

La solución propuesta para permitir a los modeladores expresar los mencionados requisitos de usabilidad sobre sus modelos consiste en realizar una extensión del metamodelo de los modelos navegacionales con aquellas entidades, relaciones y atributos que permitirán recoger esta información en tiempo de modelado.

Como se mencionó anteriormente, los modelos navegacionales están principalmente compuestos por nodos y enlaces, aunque presentan diferencias dependiendo de la metodología estudiada. Estas particularidades son un problema en el ámbito de la Ingeniería Web, ya que dificultan la interoperabilidad entre diferentes metodologías. Por ello, tal y como se mostró en el Capítulo 4, recientemente se han abordado esfuerzos como la iniciativa MDWEnet [55] orientada hacia la definición de metamodelos de referencia en los que se unifiquen los conceptos utilizados en cada metodología. De la misma forma que en el resto de metodologías, la iniciativa MDWEnet también propone un metamodelo de navegación en el que los nodos y enlaces siguen siendo los elementos principales, así que nuestra propuesta hará uso de estos elementos. Una versión simplificada de un metamodelo navegacional puede observarse en la esquina superior izquierda de la Figura 5.2.

El soporte para el tipo de requisitos mostrado en la Tabla 5.1 ha sido ofrecido extendiendo este metamodelo con los siguientes elementos (ver Figura 5.2):

Objetivo	Requisitos	Razonamiento
Definición de niveles de importancia	La información/funcionalidades en el SIW deben ser organizadas en varios niveles	No toda la información/funcionalidades en el SIW tiene la misma importancia
Distancias entre nodos	Definir distancia máxima/mínima desde el punto de entrada al SIW para cada nivel de importancia	Posibilidad de detectar nodos etiquetados como importantes demasiado alejados del punto de entrada y nodos etiquetados como menos importantes que estén excesivamente cerca
	Definir distancias máxima/mínima desde el punto de entrada al SIW para cada nodo	Posibilidad de detectar nodos poco accesibles para los usuarios
	Definir distancias entre nodos dados	Si dos nodos representan información o funcionalidades relacionadas, los modeladores probablemente desearan que ambos nodos estén cercanos
Restricciones de conectividad	Establecer conectividad directa entre nodos	Puede ser útil expresar el requisito de que dos nodos deben estar directamente conectados
	Establecer conectividad indirecta entre nodos	Puede ser útil expresar que un nodo debe ser alcanzable desde otro
Restricciones en la navegación	Forzar el paso previo por un nodo	Para cada nodo es posible definir un conjunto de nodos que deben ser previamente visitados antes de alcanzarlo
	Forzar el paso posterior por un nodo	Para cada nodo es posible definir un conjunto de nodos que deben ser visitados después de visitarlo

Tabla 5.1: Un conjunto de requisitos de usabilidad que no pueden ser expresados sobre modelos navegacionales

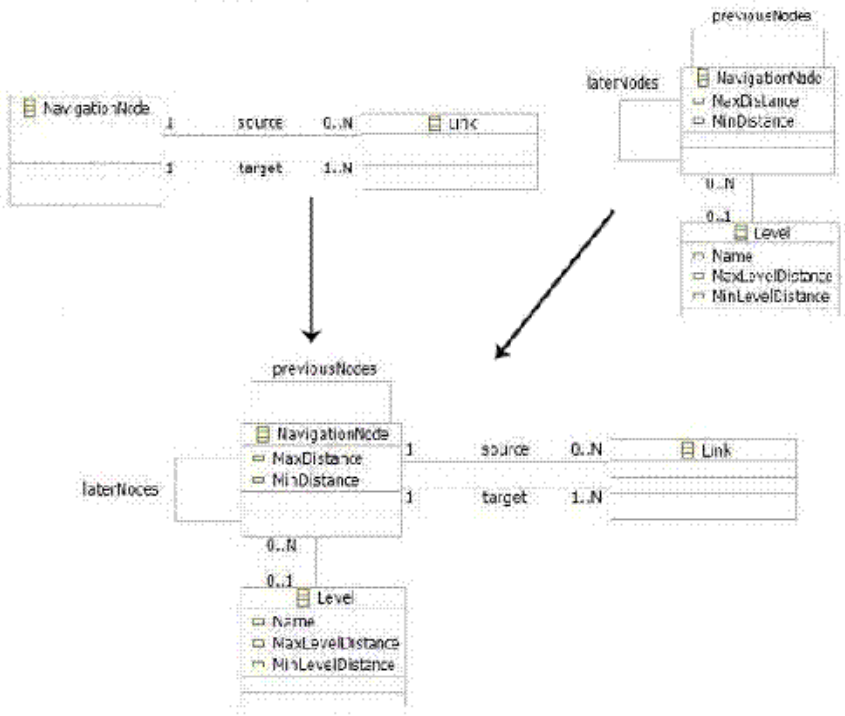

Figura 5.2: Dos opciones para extender los metamodelos navegacionales

- Añadiendo nuevos atributos. Estos atributos (llamados *MaxDistance* y *MinDistance*) son añadidos al elemento *NavigationNode* y se utilizan para dar soporte a los requisitos relacionados con las distancias entre nodos.

- Añadiendo nuevos enlaces. Concretamente, dos enlaces recursivos han sido añadidos para los nodos de navegación (llamados *previousNodes* y *laterNodes*). Estos enlaces representan, para cada nodo, una lista de nodos anteriores y posteriores que el usuario debe visitar en caso de que visite ese nodo y que son útiles para soportar los requisitos relacionados con las restricciones en la navegación.

- Añadiendo nuevas entidades. Éste es el caso de la entidad *Level* que, junto con sus atributos, es usada para soportar los requisitos relacionados con la importancia de una determinada información o funcionalidad presente en el sistema. Esta entidad representa el concepto de *importancia de un nodo* y su enlace con la entidad *NavigationNode* permite a los modeladores etiquetar cada nodo con un nivel de importancia. Cada *Level* tiene tres atributos: un nombre y dos enteros (llamados *MaxLevelDistance* y *MinLevelDistance*), los cuáles son utilizados para definir las distancias máxima y mínima entre los nodos etiquetados con un *Level* y el nodo que representa el punto de entrada al sistema.

El metamodelo obtenido después de añadir estos elementos es mostrado en la parte inferior de la Figura 5.2. Es importante resaltar que el conjunto de requisitos y la extensión

de los metamodelos navegacionales presentados son usados para ilustrar la propuesta pero, lógicamente, otros requisitos que requirieran nuevas extensiones podrían también ser descubiertos y ésta podría incluso ser utilizada sobre otros modelos usados en el desarrollo de SIW (por ejemplo, modelos de presentación) y no sólo sobre modelos navegacionales. En ese caso, sería necesario seguir la misma estrategia, consistiendo ésta en añadir aquellos atributos, entidades y relaciones que permitieran la expresión de esos nuevos requisitos sobre los modelos implicados.

Después de analizar esta extensión y explorar en mayor profundidad sus detalles técnicos, se observó que existían, al menos, dos opciones para obtener este metamodelo extendido:

1. Realizar una extensión directa de los metamodelos navegacionales propuestos por las metodologías de desarrollo de sistemas web. En este caso, el metamodelo de los modelos navegacionales sería tomado como entrada (ver esquina superior izquierda de la Figura 5.2), y todos los nuevos elementos serían añadidos sobre éste. El resultado de esta opción se muestra en la parte inferior de la Figura 5.2.

2. Crear un metamodelo independiente que agrupe aquellos elementos (atributos, entidades y relaciones) que son necesarios para dar soporte a los nuevos requisitos (ver esquina superior derecha de la Figura 5.2), los cuáles serían entonces combinados con el metamodelo navegacional original (ver esquina superior izquierda de la Figura 5.2) utilizando para ello transformaciones entre modelos.

En nuestro caso, se ha elegido la segunda posibilidad. El principal problema que presenta la primera opción es que crea un fuerte acoplamiento entre los elementos que originalmente componen los metamodelos navegacionales y los nuevos elementos añadidos sobre ellos. Este hecho podría limitar la extensibilidad de la propuesta y, además, sería más complicado para los modeladores distinguir entre los elementos del metamodelo original y los elementos añadidos con el fin de soportar los requisitos de usabilidad.

Por el contrario, en la segunda opción los metamodelos originales y las extensiones usadas para considerar características de usabilidad sobre ellos permanecen desacoplados. Ésto facilita la posible extensión de la propuesta presentada ya que para soportar nuevos requisitos únicamente sería necesario centrarse en el metamodelo que agrupa a los nuevos elementos aunque, en contrapartida, sería necesario definir la transformación con la que obtener el metamodelo destino a partir de los dos metamodelos que permanecen separados.

Este mecanismo para resolver el problema de mezclar modelos y metamodelos corresponde con un tipo especial de transformación denominado *model weaving* [197]. Las transformaciones de *model weaving* intentan resolver el problema de dado un modelo m_a que es conforme con un metamodelo M_a y un modelo m_b conforme con un metamodelo M_b, obtener un modelo integrado m_{ab} que sea conforme con un metamodelo M_{ab}, resultado de la unión de los metamodelos originales. Esta situación es similar a la presentada anteriormente: se dispone de dos metamodelos (el metamodelo navegacional original (M_a) y el metamodelo creado para dar soporte a las características de usabilidad (M_b), los cuáles son mostrados en la parte superior de la Figura 5.2) y se desea obtener

modelos navegacionales con características de usabilidad que deben ser conformes con el metamodelo mostrado en la parte inferior de la Figura 5.2.

Las transformaciones de *weaving* pueden ser especificadas en cualquier lenguaje de transformaciones como, por ejemplo, QVT [62], el estándar del OMG para este propósito en el ámbito de las estrategias de desarrollo dirigido por modelos. La Figura 5.3 muestra un fragmento de las transformaciones que permiten realizar el *weaving* entre modelos m_a y m_b (es decir, un modelo navegacional y un modelo con características de usabilidad, que son conformes con sus respectivos metamodelos mostrados en la parte superior de la Figura 5.2) para obtener el modelo m_{ab}. Estas transformaciones han sido definidas usando el lenguaje *QVT Relations*, uno de los lenguajes definidos en la especificación de QVT. En la Figura 5.3 solamente se muestra cómo los principales elementos de cada metamodelo son unidos en el metamodelo M_{ab}. Estos elementos son:

- De M_b *NavigationNode* a M_{ab} *NavigationNode* (Figura 5.3 (a)). Esta relación transfiere los nodos de navegación del modelo de usabilidad ("nn" en la Figura 5.3 (a)) al modelo destino, que a partir de ahora denominaremos *modelo de weaving* ("wnn" en la Figura 5.2), usando los valores del atributo "nn".

- De M_b *Level* a M_{ab} *Level* (Figura 5.3 (b)). De la misma forma que en la relación previa, la entidades de la metaclase *Level* en el modelo de usabilidad son transferidas al *modelo de weaving*.

- De M_a *Link* a M_{ab} *Link* (Figura 5.3 (c)). Finalmente, la entidad *Link* en el modelo navegacional es creada en el *modelo de weaving*. Esta relación selecciona un enlace en el modelo navegacional y las referencias a sus nodos fuente y destino y, a continuación, hace corresponder los nodos en el *modelo de weaving* que serán usados para crear el nuevo enlace en este modelo. Esta relación de *pattern matching* se define por medio de las variables "snnn" y "tnnn". Además, la clausula *when* es utilizada para asegurar que los dos nodos de navegación implicados en ella han sido previamente creados en el *modelo de weaving*.

Para la implementación de estas transformaciones se ha elegido la plataforma Eclipse junto con MediniQVT [67]. En la Sección 5.5 se mostrará la herramienta de soporte para la propuesta junto con un ejemplo de uso. Esta herramienta permite que una vez que los requisitos sean expresados sobre los modelos se evalúe el cumplimiento de los mismos. Para ello, se han programado diferentes funciones de validación que, además, se han combinado con el *plug in* OCL proporcionado por Eclipse y que permite obtener información sobre los modelos lo que, como se mostrará en la siguiente sección, resulta útil para el cálculo de métricas de usabilidad sobre modelos navegacionales. Si cualquier restricción es violada, los modeladores son informados de ello con el propósito de que puedan modificar sus modelos y resolver los problemas detectados.

```
(a)
top relation QualityNavigationNodetoWeaving {
  nnn: String; maxD, minD: Integer;
  checkonly domain usability nn : NavigationNode {
     Name = nnn,
     MaxDistance = maxD, MinDistance = minD };
  enforce domain weavingMM wnn : NavigationNode {
     Name = nnn,
     MaxDistance = maxD, MinDistance = minD };
}

(b)
top relation LeveltoWeaving {
   ln: String; maxD, minD: Integer;
   checkonly domain usability l:Level {
        Name = ln, MaxLevelDistance = maxD,
        MinLevelDistance = minD   };
   enforce domain weavingMM lw:Level{
        Name = ln, MaxLevelDistance = maxD,
        MinLevelDistance = minD   };
}

(c)
relation LinkToWeaving {
  snnn, tnnn: String;
  checkonly domain navigational l : Link {
    source = snn:NavigationNode { name = snnn }
    target = tnn:NavigationNode { name = tnnn }
  };
  checkonly domain weavingMM
    swnn : NavigationNode { Name = snnn };
  checkonly domain weavingMM
    twnn : NavigationNode { Name = tnnn };
  enforce domain weavingMM l2 : Link {
    source = swnn, target = twnn };
when {
   QualityNavigationNodetoWeaving(snn,swnn);
   QualityNavigationNodetoWeaving(tnn,twnn);
 }
}
```

Figura 5.3: Un fragmento de las transformaciones definidas

5.4. Métricas de usabilidad para modelos navegacionales

En el ámbito de la Ingeniería Web se han definido numerosas métricas de calidad orientadas a la medición de diferentes atributos de estos sistemas (por ejemplo, usabilidad) y a su uso para contribuir a la mejora de los mismos [129, 130]. Una revisión de la literatura realizada por Ruiz *et al.* [129] permitió descubrir un conjunto de 385 métricas definidas con el objetivo de evaluar la calidad de los artefactos software usados en el desarrollo de sistemas web. Sin embargo, como también se argumenta en estudios como el propio Ruiz *et al.* [129] o Seffah y Metzker [131], estas métricas no suelen estar definidas de forma precisa y, además, no suelen estar soportadas por ninguna herramienta o, si este soporte existe, éste es generalmente ofrecido por herramientas externas que son independientes de las usadas en el proceso de desarrollo del sistema.

El 48 % de las métricas documentadas por Ruiz *et al.* [129] están relacionadas con la medición de la usabilidad del sistema. La mejora de la usabilidad es el objetivo de la propuesta de extensión de metamodelos navegacionales presentada en este capítulo que, como beneficio adicional, puede ser complementada con el uso de métricas de usabilidad y contribuir a la solución de los problemas relacionados con el uso de métricas comentados anteriormente, como el soporte automático integrado y la falta de precisión en su definición. Por un lado, el soporte automático ofrecido por nuestra propuesta gracias al uso de la plataforma Eclipse permite el uso de OCL, lenguaje que puede ser utilizado para definir de forma precisa estas métricas. Además, la herramienta desarrollada puede servir como punto de integración para las diferentes métricas en lugar de que éstas sean evaluadas por diferentes herramientas independientes.

De entre las métricas de usabilidad documentadas en [129] se seleccionaron aquellas que cumplieran dos requisitos. El primero de ellos era que las métricas estuvieran relacionadas con la navegación en el sistema, por estar nuestra propuesta centrada en modelos navegacionales. Como además nuestro enfoque se orienta hacia la evaluación temprana de la usabilidad, se exigió como segundo requisito que las métricas elegidas deberían ser medibles en tiempo de modelado. Algunas de las métricas de usabilidad existentes como, por ejemplo, métricas orientadas a la medición de enlaces rotos en el sistema, no pueden ser evaluadas sobre modelos, ya que no existe suficiente información para este propósito en tiempo de modelado. Aún así, fue posible encontrar un conjunto de unas 50 métricas que pueden ser definidas y evaluadas sobre modelos navegacionales. En la Tabla 5.2 se muestra un resumen de estas métricas, agrupadas por el tipo de información que ofrecen, de acuerdo con los dos tipos de objetivos.

Aunque una explicación más exhaustiva sobre cada una de estas métricas puede encontrarse en [129], la Tabla 5.2 muestra que estas métricas ofrecen información cuantitativa sobre los modelos (por ejemplo, número de nodos y enlaces, profundidad y anchura del modelo, etc.) o información acerca de la conectividad entre los elementos en los modelos (por ejemplo, la relación entre el número de nodos y enlaces o el número de nodos de entrada y salida, etc.).

Una vez identificadas estas métricas, el siguiente objetivo fue su integración como parte de la herramienta de evaluación temprana de usabilidad que soporta toda la propuesta

Objetivo	Ejemplos de métricas
Información cuantitativa	Profundidad, anchura, diámetro, radio, número de nodos, número de enlaces, número de atributos y métodos, etc.
Conectividad en el modelo	Número de enlaces de entrada y salida, compactación, estrato, distancia media conectada, densidad, etc.

Tabla 5.2: Ejemplos de métricas de usabilidad para modelos navegacionales

presentada en este capítulo. A continuación, se muestran los detalles de esta herramienta y se ilustra con un ejemplo que también servirá para mostrar como los modeladores pueden definir sus propias métricas y consultas sobre los modelos creados mediante el uso de OCL.

5.5. Soporte automático y ejemplo de uso

Uno de los factores clave para el éxito de las propuestas para la gestión de usabilidad reside en la existencia de un soporte automático adecuado para ellas siendo positivo además éste pueda ser integrado en las herramientas usadas para el desarrollo de sistemas web. Por ello, en las siguientes secciones se mostrará el prototipo de la herramienta implementada para ofrecer soporte a la propuesta de evaluación temprana de usabilidad explicada en este capítulo apoyándonos en un ejemplo de uso.

5.5.1. Entorno tecnológico

Como se comentó en la Sección 2.6, la plataforma Eclipse ha sido la elegida para la implementación de herramientas en el marco de esta tesis por diferentes razones. Eclipse, a través de EMF, permite la definición sencilla de los metamodelos implicados en nuestra propuesta. Además, como se desea que diferentes *stakeholders* y no sólo los modeladores la utilicen, se hace necesario ofrecer una herramienta gráfica con una interfaz cómoda y usable que permita la creación y manipulación de modelos así como la evaluación de propiedades sobre los mismos de forma sencilla. Esta herramienta gráfica puede ser obtenida mediante el uso de GMF.

Por otro lado, la estrategia realizada para la evaluación temprana de usabilidad no está pensada para ser un esfuerzo aislado sino una iniciativa que pueda ser integrada en los procesos de desarrollo de sistemas web con el objetivo de reforzar el tratamiento dado a las tareas de evaluación de usabilidad. Las capacidades ofrecidas por Eclipse para extender las herramientas implementadas con nuevas funcionalidades así como para integrar éstas en otras herramientas existentes facilita esta tarea.

Además, algunas herramientas CAWE (*Computer-Aided Web Engineering*) como WebRatio [109] diseñadas para el soporte de metodologías de desarrollo de sistemas

web están siendo migradas de forma progresiva a esta plataforma y los esfuerzos para alcanzar un acuerdo común en la disciplina como MDWEnet también tienen a Eclipse como soporte tecnológico. La integración del soporte automático desarrollado en estas herramientas basadas en Eclipse puede ser obtenida de forma directa a través de un nuevo *plug in*. Por último, Eclipse ofrece una implementación de OCL [198] que, como comentamos anteriormente, es útil para la definición de métricas sobre los modelos navegacionales.

Una vez seleccionada esta plataforma, los pasos llevados a cabo para la implementación del soporte automático fueron los siguientes:

- Definición de los metamodelos de navegación y usabilidad (Figura 5.2) utilizando EMF.
- Implementación de los editores gráficos que permiten a los *stakeholders* la definición de modelos de requisitos y modelos navegacionales con características de usabilidad de una forma sencilla e intuitiva. Los modelos, que serán conformes a los metamodelos definidos en el paso anterior, se construirán mediante el uso de una paleta de elementos similar a la incluida en cualquier herramienta CASE. Para la implementación de estos editores se ha hecho uso del proyecto GMF.
- Definición de las transformaciones explicadas en la Sección 5.3.1.
- Definición e implementación de los chequeos que permiten la evaluación de requisitos y métricas de usabilidad (Tablas 5.1 y 5.2) , la cuál se apoya en la implementación de OCL ofrecida por la plataforma Eclipse.

Veamos a continuación la apariencia y funcionalidad de la herramienta a través de un ejemplo de uso.

5.5.2. Ejemplo de uso

En esta sección se ilustra cómo la propuesta presentada para la evaluación de atributos de usabilidad en tiempo de modelado puede ser utilizada. Supongamos un ejemplo similar al mostrado en el Capítulo 4 en el que se pretende abordar la construcción de un sistema *online* que mejore la venta de productos a través de la red, por ejemplo, la venta de libros.

En este capítulo nos centraremos en ilustrar la propuesta de evaluación temprana de usabilidad pero, en primer lugar, habría que trabajar en la identificación de los requisitos del sistema con las técnicas de metamodelado explicadas en el Capítulo 4. En la Figura 5.4 se muestra un extracto de un posible modelo de requisitos para este sistema de venta de libros *online* en el que, de forma simplificada, aparecen requisitos relacionados con la venta de libros como su visualización o proceso de compra, que no se detallan por no ser el objetivo de este capítulo. Si nos centramos en los requisitos relacionados con la usabilidad y accesibilidad del sistema, se ha establecido que éste debe cumplir con las

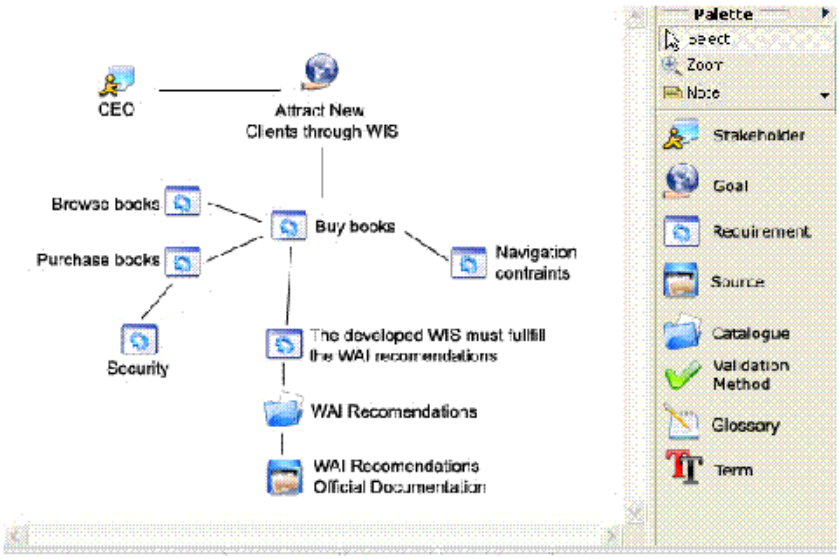

Figura 5.4: Extracto del modelo de requisitos para el ejemplo de uso

guías de accesibilidad WAI (*Web Accessibility Initiative* [128]) del W3C mientras que se han impuesto también una serie de requisitos de navegación (*Navigation constraints* en la Figura 5.4) que se utilizan para representar restricciones sobre la navegación en el sistema, que son las que pasamos a detallar a continuación.

Tras la definición de los requisitos del sistema, el siguiente paso consiste en construir los modelos conceptuales que reflejan dichos requisitos. A continuación, mostraremos cómo los requisitos relacionados con la facilidad de navegación pueden ser representados sobre los modelos navegacionales y cómo la calidad de estos modelos puede ser evaluada usando pruebas y métricas definidas sobre ellos. En la Figura 5.1 se muestra un fragmento del modelo navegacional correspondiente al ejemplo comentado, que ha sido modelado utilizando el editor gráfico implementado para la propuesta.

Como estos modelos han sido extendidos para soportar características de usabilidad, los modeladores pueden expresar sobre ellos requisitos relacionados con la navegación en el sistema al mismo tiempo que construyen sus modelos navegacionales. La Figura 5.5 muestra cómo pueden definirse algunos de estos requisitos. Supongamos que el CEO de la compañía estableció que en el sistema hay información con dos niveles de importancia definidos como *High* y *Low* en la Figura 5.5, ya que el sistema permite tanto la compra de libros (que es su finalidad principal) como la compra de películas correspondientes a adaptaciones de libros (siendo ésta una funcionalidad secundaria para el sistema). Los nodos relacionados con la venta de libros han sido etiquetados con el nivel *High* mientras que los nodos relacionados con la venta de películas se han etiquetado con el nivel *Low*. Además, ciertas distancias máximas y mínimas han sido definidas para estos nodos. Como se observa en la parte inferior de la Figura 5.5, se ha definido una distancia máxima de un *click* entre los nodos etiquetados con el nivel *High* y el punto de entrada al sistema.

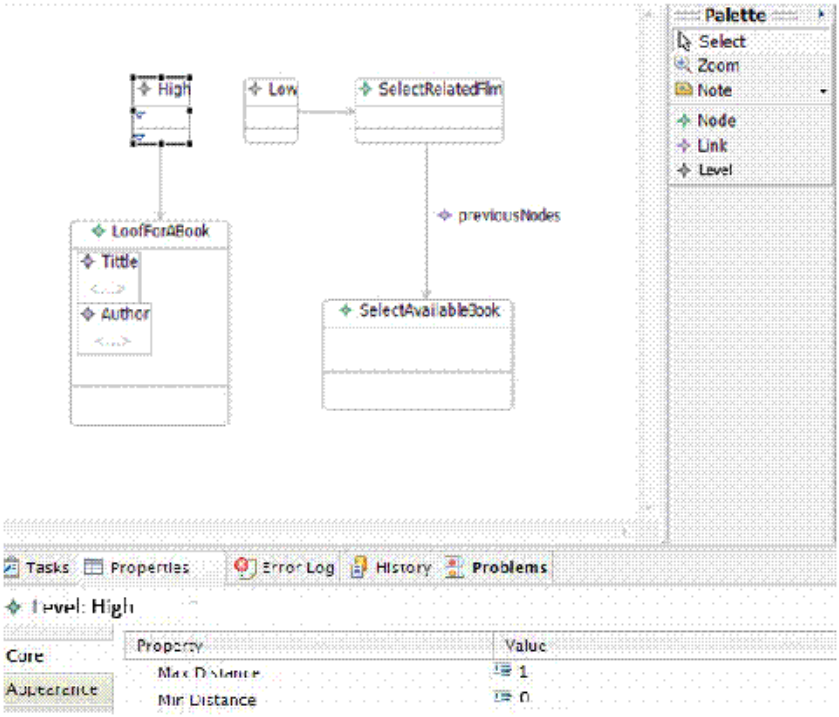

Figura 5.5: Añadiendo restricciones de usabilidad a un modelo navegacional

Después de la definición de estos requisitos de usabilidad, los modeladores pueden ejecutar las pruebas implementadas para asegurar que estos requisitos son cumplidos por los modelos creados. Además, los modeladores también tendrán a su disposición la implementación del conjunto de métricas mencionadas en la Sección 5.4 lo que les permitirá estudiar la bondad de sus modelos. La información obtenida tras la ejecución de estas pruebas permitirá a los modeladores obtener información valiosa con la que evaluar la calidad de sus modelos y que podrán usar para mejorarlos, en caso de que sea necesario.

Adicionalmente, el *plug in* para OCL permite a los modeladores definir sus propias métricas y restricciones sobre los modelos creados. La Figura 5.6 muestra algunos ejemplos sencillos de estas métricas y restricciones. La Figura 5.6(a) corresponde a un ejemplo del tipo de métricas mostradas en la Tabla 5.2. Concretamente, en este caso es una métrica cuantitativa que permite contar el número de nodos en el modelo. La Figura 5.6(b) es útil para ilustrar el tipo de métricas que un modelador puede definir y ejecutar sobre sus modelos. En este caso, la métrica es definida sobre el modelo de la Figura 5.5 y selecciona aquellos nodos etiquetados con el nivel de importancia *High*. Finalmente, es importante resaltar que los modeladores pueden definir no sólo métricas sino también restricciones sobre sus modelos. Por ejemplo, la Figura 5.6(c) ilustra una restricción que establece que todos los enlaces en el modelo deben tener nombres diferentes. Estas métricas y restricciones pueden ser incorporadas a la herramienta de forma que los modeladores simplemente tengan que seleccionarlas para obtener información que pueden usar para evaluar y mejorar sus modelos. El soporte ofrecido para este tipo de restricciones así

```
(a) self.nodes->size()
(b) self.nodes->select(n:Node | n.level.Name = "High")
(c) self.links->isUnique(l: link | l.name)
```

Figura 5.6: Ejemplos de métricas y restricciones definidas en OCL

como para otras soportadas formalmente será mostrado en más detalle en el Capítulo 6.

5.6. Resumen

A modo de resumen, en la Figura 5.7 (extraída de [22]) se muestra una visión general de la propuesta utilizando la notación SPEM. Como se aprecia, el proceso comienza con la extracción sistemática de requisitos, utilizando para ello las técnicas de metamodelado explicadas en el Capítulo 4 y haciendo énfasis, en este caso, en la elicitación temprana de requisitos de calidad como los relacionados con la usabilidad del sistema. Después de esta etapa, se diseñan los modelos conceptuales que reflejan estos requisitos. Este capítulo se ha centrado en los modelos navegacionales usados en el diseño de sistemas web, los cuáles son construidos junto con el modelo de usabilidad que ofrece a los modeladores capacidad para expresar este tipo de requisitos y que, hasta el momento, no podían ser definidos sobre los modelos navegacionales. A continuación, los requisitos de usabilidad pueden ser comprobados y diferentes métricas de calidad pueden ser calculadas sobre los modelos navegacionales con el fin de obtener información que ayude a mejorar la calidad de los mismos. Todo este *feedback* servirá para, si es necesario, mejorar los modelos del sistema. Además, en el siguiente capítulo se verá la forma en la que se pueden aplicar técnicas para la verificación de propiedades sobre estos modelos con el objetivo de seguir mejorando sus características antes de que sean usados para la implementación del sistema.

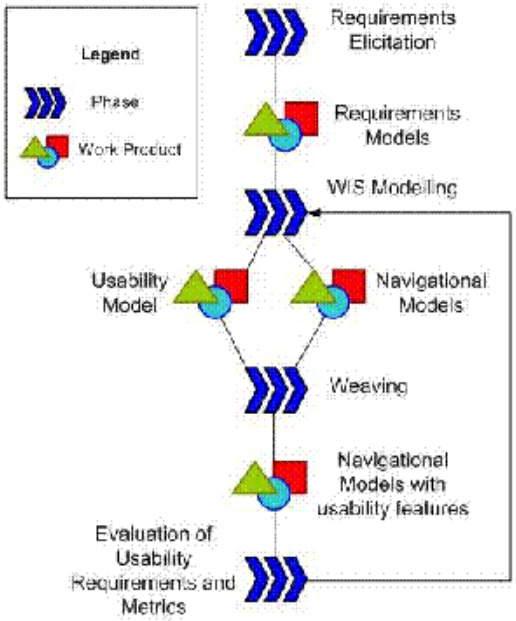

Figura 5.7: Visión general de la propuesta

Capítulo 6

Análisis preciso de modelos en Ingeniería Web

6.1. Introducción

Las propuestas *model driven* han convertido a los modelos conceptuales en artefactos críticos de los que depende gran parte del éxito de los proyectos de desarrollo software. Este hecho obliga a dedicar gran atención a la construcción de estos modelos y hace aumentar la importancia de áreas como el análisis de modelos, una línea de trabajo abierta que requiere de mayor atención por parte de la comunidad de desarrollo dirigido por modelos [11, 76].

En este capítulo resumiremos los detalles de una estrategia de verificación precisa de modelos realizada en el marco de esta tesis y que está orientada a la detección de errores y chequeo de propiedades sobre modelos conceptuales mediante el uso de métodos formales. Esta estrategia, que usa como soporte al lenguaje Maude, ha sido aplicada en el ámbito de la Ingeniería Web y presentada en [31, 32].

6.2. Verificación de modelos en SIW: primera aproximación

Como se mencionó en la Sección 2.4, las metodologías surgidas en Ingeniería Web proponen el uso de diferentes modelos para llevar a cabo el desarrollo del sistema. Además, el uso de modelos es cada vez más intensivo debido a la aparición de la denominada *Model Driven Web Engineering*, que pretende aprovechar las ventajas ofrecidas por el desarrollo dirigido por modelos en la construcción de sistemas web.

Tradicionalmente, las metodologías de desarrollo surgidas en Ingeniería Web han estado centradas, sobre todo, en los aspectos relativos al diseño del sistema dejando más de lado otras áreas del desarrollo relacionadas, por ejemplo, con la gestión de requisitos, tal y como se mostró en el Capítulo 4, o el análisis y verificación de sitios web [139]. Hace algunos años, uno de los enfoques surgidos para solucionar estas carencias consistió

en el uso de métodos formales para el análisis y verificación de SIW. De esta forma, comenzaron a aparecer trabajos orientados al análisis de sitios web como [139, 141, 142, 143]. Al mismo tiempo, y correspondiendo con el inicio de esta tesis doctoral, el Grupo de Investigación en Ingeniería del Software de la Universidad de Murcia también inició trabajos en esta línea, desarrollando una estrategia para la verificación formal de los modelos usados en el desarrollo de SIW así como para la demostración de propiedades sobre los mismos [31, 32]. Este tipo de técnicas habían sido previamente aplicadas con éxito en el desarrollo de sistemas software tradicionales [199, 200].

La estrategia de verificación formal desarrollada se basa en la especificación algebraica de los metamodelos de los modelos usados durante el desarrollo de un SIW y en la implementación de propiedades a comprobar sobre estos modelos. Para llevar a cabo esta especificación algebraica se hizo uso del lenguaje Maude (ver Sección 2.5). Con la formalización del metamodelo de un diagrama se consiguen una serie de ventajas como:

- Modelos más precisos, al ser necesario especificarlos sin ambigüedades en lenguajes basados en formalismos matemáticos.
- Posibilidad de verificar formalmente si los modelos cumplen o no determinadas propiedades.
- Posibilidad de realizar transformaciones precisas entre modelos.

Para la aplicación y validación de esta estrategia se usaron los modelos de navegación extendidos usados por la metodología MIDAS, un *framework* para el desarrollo de SIW en el que se propone un proceso de desarrollo integrado en una arquitectura dirigida por modelos [108]. En la Figura 6.1 se muestra un ejemplo de este tipo de modelo, correspondiente al modelado de un sistema de venta de billetes de avión. Los modelos de navegación extendidos se definen en base a la descomposición del sistema en unidades significativas, llamadas *fragmentos*, e *hiperenlaces* entre dichos fragmentos. Los fragmentos pueden ser de dos tipos: estructurales, que representan una unidad de información que se va a mostrar en la web de forma agrupada, y funcionales, que muestran, además de la información que debe ser mostrada, otro tipo de información o funcionalidades que permiten representar la interacción del usuario con el SIW. Cada uno de estos fragmentos se representa mediante clases estereotipadas con <<SS>> (*Structural Slice*) y <<FS>> (*Functional Slice*).

Como puede apreciarse en la Figura 6.1, en estos diagramas se modelan las rutas que un usuario puede seguir durante su navegación en el sistema. Además, y como particularidad de la metodología MIDAS, cada ruta es etiquetada con un nombre y se asocia con el conjunto de nodos y enlaces que atraviesa un usuario para conseguir llevar a cabo una tarea determinada (por ejemplo, la ruta para comprar un billete de avión en la Figura 6.1 es etiquetada con las siglas BT).

Es importante destacar que aunque un modelo sea correcto desde el punto de vista sintáctico, todavía puede contener errores y ambigüedades desde el punto de vista de su semántica dinámica o también con respecto a las reglas de buena formación (*well-formedness rules*) de su metamodelo o, incluso, con respecto a propiedades específicas del

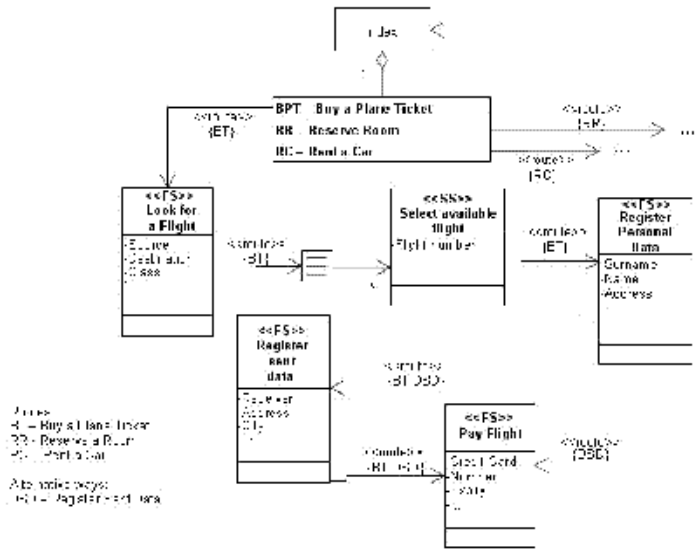

Figura 6.1: Un ejemplo de modelo de navegación extendido

propio dominio al que pertenece o relativas al sistema concreto que se está desarrollando. En el caso concreto de los modelos de navegación extendidos, algunas propiedades en este sentido están relacionadas con la existencia de rutas inconexas, rutas que no tienen un fin válido o ciclos en el modelo que pueden ser indicativos de un mal diseño.

Nuestra propuesta de análisis preciso se inició con la verificación de este tipo de propiedades. Para ello, en primer lugar, fue necesario obtener la especificación algebraica correspondiente al metamodelo de estos modelos y al conjunto de propiedades que sería interesante verificar sobre los mismos.

En la Figura 6.2 se muestra una vista simplificada de este metamodelo. De forma similar a los modelos de navegación propuestos por otras metodologías, los principales elementos que lo componen son los nodos y los enlaces entre ellos, junto con otras estructuras navegacionales como menús y, en este caso, la rutas de navegación. Estas estructuras han sido formalizadas para conseguir una especificación precisa, con base matemática, de los modelos correspondientes. En la Figura 6.3 se muestra un extracto de esta formalización que, en este caso, corresponde a los módulos utilizados para especificar las metaclases correspondientes a nodos y enlaces, así como para el diagrama de navegación. Todos los detalles relativos a esta formalización pueden encontrarse en [31, 32].

Una vez que estos metamodelos estaban formalizados, se trabajó en la implementación de las propiedades mencionadas anteriormente. Por ejemplo, en la Figura 6.4 se muestra la implementación de una propiedad (llamada *testValidRoute*) para verificar que todas las rutas en el modelo son correctas y el resultado de su ejecución sobre un modelo con errores. Supongamos que se ha introducido un error en el modelo mostrado en la Figura 6.1 y que, cuando una ruta etiquetada con el valor BT llega al nodo de nombre *Look for a Flight*, se elimina la existencia de un enlace etiquetado con BT. Teniendo en cuenta

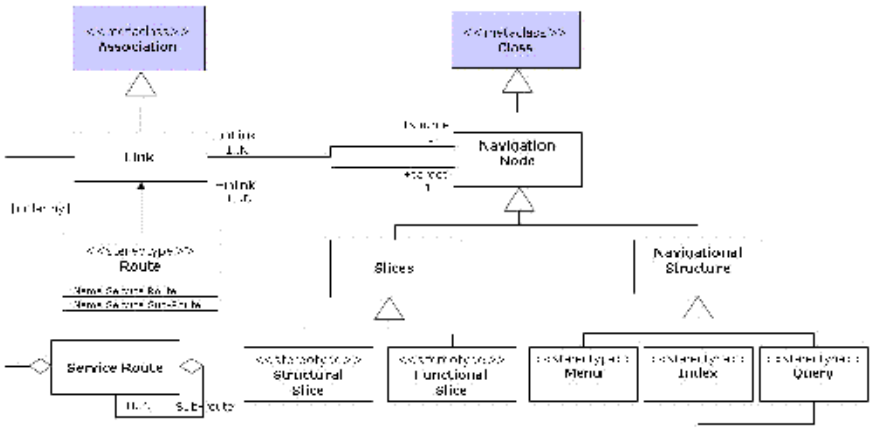

Figura 6.2: Versión simplificada del metamodelo del modelo de navegación extendido

que esta ruta no supone un fin válido para la ruta marcada por BT, la ejecución de la propiedad sobre Maude ejecutando la operación *testValidRoute* sería la que aparece en la parte inferior de la Figura 6.4.

6.3. Ofreciendo soporte automático al uso de lenguajes formales

La estrategia presentada en la sección anterior ofrece ventajas como la precisión en la definición de metamodelos y modelos o la posibilidad de verificar los mismos frente a un conjunto de propiedades. Sin embargo, el uso de la misma únicamente con los conceptos presentados en la Sección 6.2, presenta inconvenientes relacionados con la usabilidad de la propuesta por parte de usuarios que no estén familiarizados con el uso de este tipo de técnicas precisas.

Como mencionamos en la Sección 2.4.3, la dificultad de uso por parte de usuarios no expertos, la falta de soporte automático o soportes automáticos pobres y ofrecidos por herramientas independientes de las usadas en los procesos de desarrollo son críticas tradicionales realizadas con respecto a la utilización de métodos formales. Por ello, se destaca como un factor clave para aumentar su utilidad el desarrollo de herramientas de soporte y la integración de éstas en entornos industriales de desarrollo como, por ejemplo, Eclipse [148].

Siguiendo estas recomendaciones, a continuación mostraremos una arquitectura diseñada para ofrecer soporte automático a la estrategia presentada en la Sección 6.2. Esta arquitectura está siendo implementada en la plataforma Eclipse, hecho que permite su integración con las propuestas mostradas en los capítulos previos, poniendo de esta forma a disposición de los modeladores nuevas capacidades orientadas a la mejora de sus

```
(fmod NAVIGATIONNODE is
   sort NavigationNode .
   pr NAVIGATIONNODENAME .
   pr ROUTELIST .
   op navigationNode : NavigationNodeName RouteList
                       -> NavigationNode [ ctor ] .
   ...
   *** Returns true, if the node is a valid end for the route.
   op    isValidEnd : NavigationNode Route -> Bool .
   ...
endfm)

(fmod LINK is
   sort Link .
   pr LINKEND .
   pr LINKNAME .
   pr ROUTELIST .
   op link : LinkName LinkEnd LinkEnd RouteList
                     -> Link [ ctor ] .
   op getLinkName : Link -> LinkName .
   op getLinkEndSource : Link -> LinkEnd .
  ...
endfm)

(fmod NAVIGATIONDIAG is
   sort NavigationDiag
   pr NAVIGATIONNODELIST .
   pr LINKLIST .
   op navigationDiag : NavigationNodeList LinkList ->
                       NavigationDiag [ ctor ] .
   op getNavigationNodeList : NavigationDiag ->
                       NavigationNodeList .
   op getLinkList : NavigationDiag -> LinkList .
   ...
endfm)
```

Figura 6.3: Extracto de la formalización en Maude del metamodelo de la Figura 6.2

```
(fmod VERIFICATIONS is
   pr UTILSDNAV .
   pr STRING .
   op testValidRoute : NavigationDiag NavigationNodeList
          ServiceRouteList -> String .
   var NN : NavigationNode .
   var NNL : NavigationNodeList .
   var R : ServiceRouteList .
   var ND : NavigationDiag .

   *** This route finishes in a valid final node. So, this
   partial route does not have errors.
   eq testValidRoute (ND, nullNavigationNodeList, R) = "" .
   eq testValidRoute (ND, NN NNL, R) =
      if selectLinks(nextLinks(ND, NN), route(R,noRouteEnd))
    == nullLinkList and not(isValidEnd(NN, route(R,noRouteEnd)))

     then       "ERROR, the route " + ... + " which is not a
                 valid final node."
     else testValidRoute(ND,getTargetNodes(ND,
                         selectLinks(nextLinks(ND, NN),
     route(R,noRouteEnd))) NNL, R)
      fi .
endfm)

reduce in ejemploNavigD :
testValidRoute(reserveND,index,'BT)
result String :   "ERROR, the route BT finishes in the
slice lookForFlight which is not a valid final node. "
```

Figura 6.4: Ejemplo de propiedad implementada y resultado de su ejecución

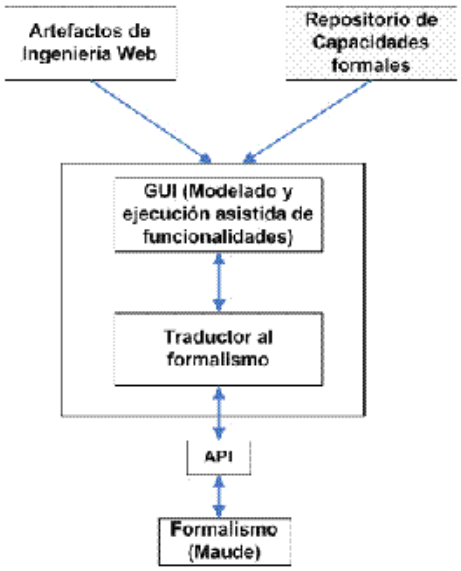

Figura 6.5: Visión general de la arquitectura propuesta

artefactos software.

6.3.1. Visión general de la arquitectura

En el diseño del soporte automático para la propuesta se han considerado varios requisitos. Como se mencionó previamente, las técnicas formales suelen ser ajenas a los modeladores web. Por tanto, un requisito importante es que la arquitectura diseñada oculte a los modeladores los detalles relacionados con el formalismo subyacente. Con este objetivo, la herramienta propuesta debe ofrecer una interfaz amigable que permita la ejecución asistida de las funcionalidades implementadas (o que puedan añadirse en el futuro). En este sentido también resultan importantes las capacidades de extensión de la herramienta, ya que el número de funcionalidades formalmente implementadas no tiene que ser estático, sino que debe ser posible añadir nuevas utilidades si así se estima conveniente.

Por otro lado, la arquitectura debe facilitar su integración con las propuestas realizadas en los capítulos previos, ya que la estrategia no está pensada para ser un esfuerzo aislado. Este hecho influyó en la elección de Eclipse como plataforma tecnológica.

Considerando estos requisitos, se diseñó la arquitectura mostrada en la Figura 6.5. Los principales elementos que la componen son:

- Artefactos de Ingeniería Web. Con este elemento se representan los diferentes artefactos implicados en el proceso de desarrollo de SIW (por ejemplo, modelos conceptuales) sobre los que tiene sentido la aplicación de técnicas formales.
- Repositorio de capacidades formales. Este concepto representa al conjunto de utilidades formalmente especificadas que se encuentra a disposición de los modeladores

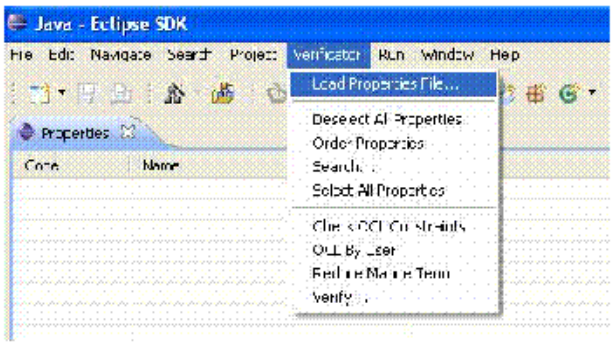

Figura 6.6: Menú para tratar con la aplicación de técnicas formales

para su aplicación sobre sus artefactos software. Ejemplos de utilidades en este repositorio pueden ser pruebas o propiedades a verificar sobre los modelos, como las mostradas en la Sección 6.2.

- Formalismo. Representa al lenguaje destino sobre el que se especifican y ejecutan las capacidades formales (en este caso, Maude).
- Interfaz Gráfica de Usuario (GUI). Los modeladores utilizarían esta interfaz para seleccionar las propiedades existentes en el repositorio que desean ejecutar sobre sus artefactos software, recibiendo a través de ella información acerca de los resultados obtenidos tras esta ejecución.
- Traductor al formalismo. Este módulo transformaría los modelos creados por los usuarios de la herramienta (por ejemplo, modelos navegacionales) a especificaciones formales en el lenguaje correspondiente (como las mostradas en la Figura 6.3). Si se ofreciera soporte a varios formalismos, podrían introducirse diferentes traductores. Este proceso de traducción se realiza automáticamente, evitando que los usuarios tengan que conocer cualquier detalle acerca del formalismo destino.
- API de comunicación con el formalismo. Este módulo permite la comunicación con el formalismo elegido (Maude en este caso) y permitiría invocar y ejecutar sobre éste las capacidades formales implementadas.

Una vez identificados estos elementos, veamos cómo se ha implementado cada uno de ellos en el ámbito de Eclipse. Para la manipulación de modelos y metamodelos, el proceso fue explicado en capítulos previos. Los metamodelos pueden ser definidos en Eclipse a través de Ecore y, para trabajar con ellos de forma visual, se pueden construir editores gráficos mediante el uso del proyecto GMF. De esta forma, los modelos se manipularían usando un editor gráfico como el mostrado en la Figura 5.1. Por el momento, la propuesta se centra en modelos navegacionales.

Además de para la manipulación de modelos, esta interfaz gráfica permitirá a los modeladores de SIW usar métodos formales sin necesidad de conocer los detalles del formalismo. Para ello, el editor gráfico ha sido extendido con un menú de nuevas opciones que se

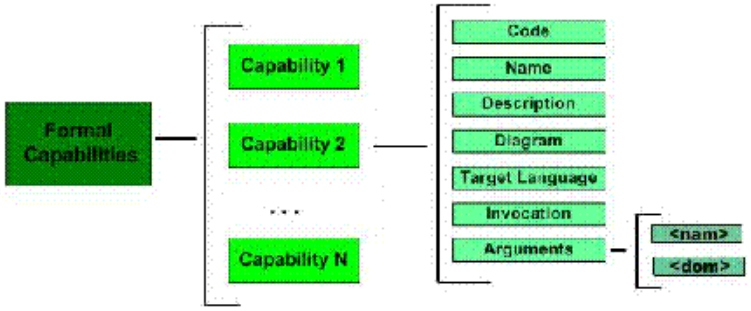

Figura 6.7: Estructura del repositorio XML

muestra en la Figura 6.6 y que ofrece funcionalidades como, por ejemplo, cargar un repositorio de capacidades formales o aplicar las elegidas por el usuario sobre los modelos construidos. Este repositorio ha sido implementado a través de un fichero XML. Cada elemento en este fichero representa una capacidad formal (ver Figura 6.7) y se describe utilizando un conjunto de atributos como:

- *Code*: usado como un identificador.
- *Name*: breve descripción asociada con cada capacidad.
- *Description*: ofrece una descripción más detallada que sirve a los modeladores para entender el propósito de una determinada capacidad.
- *Args*: si la ejecución de una funcionalidad necesita argumentos, serían almacenados en este atributo.
- *Diagram*: modelo sobre el que se aplica una capacidad (por ejemplo, modelo navegacional).
- *Target Language*: lenguaje formal en el que se especifica la capacidad (Maude en este caso)
- *Invocation*: código usado para invocar la capacidad en el formalismo elegido.

Gracias a esta representación, añadir nuevas capacidades al repositorio resulta sencillo ya que solamente es necesario añadir una nueva entrada al fichero y rellenar los atributos correspondientes. Además, este formato de representación es fácilmente integrable en Eclipse y transparente a los usuarios ya que, como se mostró en Figura 6.6, la herramienta ofrece opciones para cargar estos ficheros y mostrar las propiedades a través de sus descripciones. En la Figura 6.8 se muestra una captura en la que se puede observar la forma en la que el modelador percibe las capacidades a su disposición y sobre la que únicamente necesita marcar aquellas en las que está interesado.

En cuanto a los módulos de transformación de modelos, tienen el objetivo de acercar dos espacios tecnológicos separados [201]. Por una lado, los modeladores de SIW trabajan

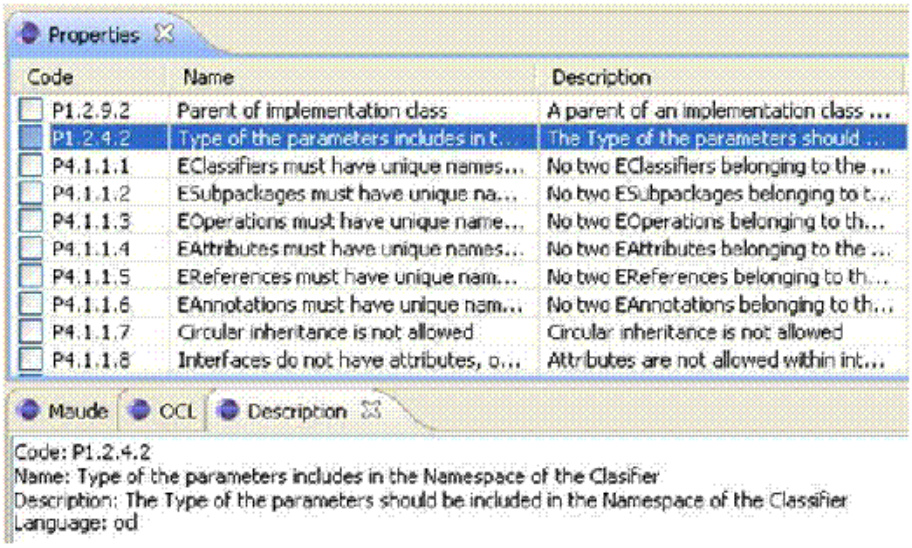

Figura 6.8: Interfaz gráfica para la selección de capacidades formales

con conceptos como páginas, nodos o enlaces entre modelos. Por otro lado, el uso de técnicas formales requiere la especificación de estos modelos en un lenguaje formal. Por tanto, es necesario construir los puentes entre estos espacios ofreciendo mecanismos que permitan transformar los modelos a su correspondiente especificación formal.

Para conseguir este propósito se han construido traductores mediante transformaciones modelo a código que toman como entrada los modelos especificados en Ecore y obtienen su representación en Maude. El lenguaje elegido para implementar esta transformación ha sido MOFScript [154], creándose un proceso de traducción completamente transparente a los usuarios de la herramienta.

Una vez que los modelos y metamodelos han sido trasladados al formalismo y el modelador ha seleccionado un conjunto de capacidades formales a aplicar sobre ellos, es necesario llevar a cabo su ejecución. En este caso, esto implica establecer una comunicación entre la herramienta diseñada y Maude, algo que se ha conseguido a través del *plug in Maude Development Tool* [202], el cuál está integrando en Eclipse y ofrece comandos que permiten intercambiar información con el entorno de ejecución de Maude. La especificaciones formales obtenidas tras el proceso de traducción y las capacidades formadas seleccionadas son enviadas a Maude, donde se ejecutan y se obtienen resultados que son mostrados a los modeladores y que pueden ser usados para mejorar sus modelos conceptuales durante la fase de modelado, evitando que errores o diseños poco eficientes se propaguen hacia fases más avanzadas del ciclo de desarrollo.

Capítulo 7

Conclusiones y vías futuras

7.1. Análisis de la consecución de objetivos

A modo de resumen, en esta sección se muestran las contribuciones realizadas en el marco de esta tesis doctoral y su relación con los objetivos definidos en la Sección 1.2 del Capítulo 1.

- **Objetivo 1:** analizar la incidencia que las fases relacionadas con la elicitación de requisitos y el modelado conceptual del sistema tienen en el éxito final de proyectos de desarrollo software. En este análisis se incluirá el estudio de los paradigmas de desarrollo dirigido por modelos, analizando tanto las características como las técnicas y herramientas promovidas por estas propuestas.

    - En el Capítulo 2 se mostró una descripción del estado arte en aquellas áreas relacionadas con esta tesis. En la misma, se destacaron las causas que con frecuencia motivan problemas en los proyectos de desarrollo software y se analizaron las características y retos a los que se enfrenta el paradigma de desarrollo dirigido por modelos. La influencia que este paradigma está teniendo en la gestión de requisitos fue analizada en el Capítulo 4 en el que se ofreció una revisión de la literatura que permitió descubrir las propuestas existentes para el metamodelado de requisitos en el marco de MDD. Las líneas de investigación abiertas en Ingeniería Web también fueron destacadas en el Capítulo 2 y se propusieron aportaciones sobre ellas con respecto a la gestión de requisitos (Sección 4.5.3), evaluación de usabilidad (Capítulo 5) y análisis de modelos (Capítulo 6). Además, nuestra revisión de la literatura sobre consistencia en UML [8] aportó una visión del estado del arte con respecto a este tipo de problemas y permitió establecer las bases para abordar líneas de investigación relacionadas con consistencia entre modelos tanto en el ámbito de Ingeniería Web como en otros ámbitos de desarrollo.

- **Objetivo 2:** desarrollar un conjunto de contribuciones alineadas con el paradigma MDE que, integradas en un método de desarrollo, contribuyan a mitigar las

carencias detectadas durante el análisis del estado del arte.

- Las contribuciones realizadas para cumplir con este objetivo constituyen el núcleo fundamental de esta tesis doctoral y se encuentran detalladas en los capítulos 4, 5 y 6 e integradas en el método de mejora propuesto en el Capítulo 3. En el Capítulo 4 se mostró URM, la propuesta realizada para el metamodelado de requisitos, que posteriormente fue extendida tanto para ofrecer soporte a la elicitación de medidas asociadas con los requisitos (Sección 4.5) como para dar soporte a la definición de requisitos de seguridad (Sección 4.7) además de ser integrada en el ámbito de la Ingeniería Web (Sección 4.5.3). En el Capítulo 5 se mostró una propuesta para la definición de requisitos y métricas relacionadas con la usabilidad del software en fases tempranas de su desarrollo mientras que en el Capítulo 6 se han mostrado técnicas para el análisis preciso de modelos conceptuales ilustradas con los modelos navegacionales usados por las metodologías de Ingeniería Web.

- **Objetivo 3:** diseñar e implementar los prototipos software que den soporte a las contribuciones comentadas en el Objetivo 2.

  - El soporte automático diseñado para cada una de las contribuciones obtenidas ha sido mostrado junto a la explicación de las mismas. La plataforma Eclipse ha sido la seleccionada para diseñar los prototipos de estas herramientas, que pueden observarse en los Capítulos 4, 5 y 6.

- **Objetivo 4:** mostrar la aplicación de las contribuciones obtenidas para la consecución del Objetivo 2 utilizando como campo de estudio la disciplina de Ingeniería Web.

  - Aunque las contribuciones realizadas son de aplicación en múltiples ámbitos, la Ingeniería Web ha sido el campo de estudio seleccionado por los motivos indicados en el Capítulo 1. Los ejemplos mostrados en los capítulos 4, 5 y 6 ilustran las contribuciones aportadas en esta tesis con ejemplos de este área relacionados con la elicitación de requisitos y medidas (Sección 4.6), evaluación temprana de requisitos y métricas de usabilidad (Sección 5.5) y análisis de modelos mediante el uso de técnicas de reescritura (Sección 6.2).

A continuación, y de forma resumida, se destacan los resultados obtenidos tras la realización de esta tesis, que se acompañan junto con las publicaciones más relevantes:

- Una propuesta para el metamodelado de requisitos (URM) que formaliza los conceptos implicados en el proceso de elicitación de requisitos y las relaciones entre ellos. Esta propuesta ha servido para reforzar el tratamiento que reciben los requisitos en el desarrollo de software además de como base para la realización otras contribuciones que detallamos a continuación [22, 25].

- Una propuesta para la conexión del metamodelo de requisitos mencionado anteriormente con metamodelos de medición de software, con el objetivo de ligar los requisitos elicitados con medidas que ayuden a comprobar su cumplimiento en el sistema desarrollado. Esta propuesta viene acompañada de una notación propia y ha sido integrada en el ámbito de la Ingeniería Web para mostrar su uso [25, 27].
- Una propuesta para la elicitación temprana de requisitos de seguridad que usa como punto de partida el metamodelo de requisitos desarrollado, que ha sido extendido con características específicas del campo de la seguridad obteniendo un metamodelo de requisitos de seguridad que sirve como base para la construcción de una arquitectura MDE para la generación de artefactos software, como código, relacionados con la seguridad del sistema [29, 30].
- Una propuesta para la consideración y evaluación de requisitos relacionados con la usabilidad del sistema desde fases tempranas de su desarrollo basada de nuevo en las prácticas MDE [22, 28].
- Un conjunto de técnicas orientadas al análisis de modelos conceptuales que, en este caso, están basadas en el uso de métodos formales [31, 32]. Concretamente, se ofrecen técnicas para la verificación de modelos que han sido aplicadas en el ámbito de la Ingeniería Web. Además, se trabajó en la elaboración de una revisión sistemática [8] que ha permitido conocer el estado del arte relativo a la consistencia entre modelos y que más tarde sirvió para realizar una propuesta inicial orientada al análisis de consistencia aplicada sobre modelos de Ingeniería Web [33].

7.2. Contraste de resultados de la investigación

Durante el desarrollo de esta tesis doctoral se han publicado y discutido varios trabajos en diversos foros científicos. La Tabla 7.1 muestra un resumen numérico de estas publicaciones, todas ellas sometidas a procesos de revisión. Además, la tabla incluye información acerca de los Proyectos Fin de Carrera realizados en líneas de investigación relacionadas con esta tesis y en cuya dirección ha participado el doctorando.

En las siguientes secciones se muestra cada uno de estos trabajos desglosados por categorías.

7.2.1. Artículos en revistas internacionales indexadas en JCR

A continuación se muestran los artículos en revistas listados en los índices de impacto ISI/JCR (*Journal Citation Report*), junto con el índice de impacto de las mismas:

- Molina, F. y Toval, A (2009). Integrating usability requirements that can be evaluated in design time into Model Driven Engineering of Web Information Systems. Journal of Advances in Engineering Software, Vol. 40, Issue 12, pp. 1306-1317. **Factor de impacto en 2008: 1.188**

Tipo de Contribución	Total
Artículos en revistas internacionales ISI/JCR	4
Capítulos en libros internacionales	1
Capítulos en libros nacionales	1
Congresos y talleres internacionales	7
Congresos y talleres nacionales	3
Proyectos Fin de Carrera	3
Proyectos de transferencia tecnológica	1
Informes Técnicos	1
Número total de contribuciones	21

Tabla 7.1: Número de publicaciones y trabajos relacionados con la tesis organizadas por tipos

- Molina, F., Pardillo, J., Cachero, C. y Toval, A (2010). An MDE Modelling Framework for Measurable Goal-oriented Requirements. International Journal of Intelligent Systems. Aceptado para su publicación en el Special Issue 'Goal-Driven Requirements Engineering'. **Factor de impacto en 2008: 0.860**

- Lucas, F.J., Molina, F. y Toval, A (2009). A Systematic Review of UML Model Consistency Management. Information and Software Technology, Vol. 51, Issue 12, pp. 1631-1645. **Factor de impacto en 2008: 1.2**

- Sánchez, O, Molina, F., García-Molina, J. y Toval, A (2010). ModelSec: a generative architecture for model-driven security. Journal of Universal Computer Science. Aceptado para su publicación en el Special Issue 'Security in Information Systems: New Advances and Tendencies'. **Factor de impacto en 2008: 0.488**

7.2.2. Capítulos en libros internacionales

- F. Molina, F.J. Lucas, A. Toval, J.M. Vara, E. Marcos (2008). Towards Quality Web Information Systems through Precise Model Driven Development. En Handbook of Research on Web Information Systems, pp. 317-355. Idea Group Publishing, 2008. ISBN: 978-1-59904-847-5.

7.2.3. Congresos y talleres internacionales

7.2.3.1. Publicados LNCS *proceedings*

- Molina, F., Pardillo, J., Toval, A. Modelling Web-based Systems Requirements using WRM. 2nd Int. Workshop on Web Usability and Accessibility (IWWUA'08) conjunto con la 9th Int. Conf. on Web Information Systems Engineering (WISE'08). LNCS Volume 5176, pp. 122-131. Auckland, Nueva Zelanda, 2008

- Pardillo, J., Molina, F., Cachero, C., Toval, A. A UML Profile for Modelling Measurable Requirements. 4th Int. Workshop on Foundations and Practices of UML (FPUML'08), conjunto con la 27th International Conference on Conceptual Modelling (ER 2008). LNCS 5232, pp. 123-132. Barcelona, 2008

- Molina, F., Toval, A. A generic approach to improve navigational model usability based upon requirements and metrics. 1st Int. Workshop on Web Usability and Accessibility (IWWUA'07) conjunto con la 8th Int. Conf. on Web Information Systems Engineering (WISE'07). LNCS Volume 4832, pp. 511-516. Nancy, Francia, 2007

7.2.3.2. Publicados en otros *proceedings*

- F.J. Lucas, F. Molina, A. Toval. (2009). A Rule-based Approach for Managing Semantic Inconsistencies in Web Information Systems Development. 5th International Workshop on Automated Specification and Verification of Web Systems (WWV'09). Austria. Julio, 2009.

- O. Sánchez, F. Molina, J. García-Molina, A. Toval (2009). A model driven approach for generating code from security requirements. Proceedings del International Workshop on Security In Information Systems (WOSIS'09), pp, 119-126. Milan (Italia), Mayo, 2009.

- Molina, F., Cachero, C., Pardillo, J., Toval, A. Towards a Requirements-Aware Common Web Engineering Metamodel. Latinoamerican Conference of Web Engineering (LAWEB'08), pp. 75-82. IEEE Digital Library. Brasil, 2008.

- Lucas, F.J., Molina F., Toval, A., De Castro, M.V., Cáceres, P., Marcos, E. Precise WIS Development. International Conference on Web Engineering (ICWE '06), pp. 71-76. Menlo Park, California (USA). ACM Proceedings. ISBN: 1-59593-352-2.

7.2.4. Capítulos en libros nacionales

- F. Molina, C. Cachero, J. Pardillo, A. Toval (2010). Metamodelado de requisitos medibles. En Calidad de Producto y Proceso Software, Capítulo 20. Editores: C. Calero and Mª. Ángeles Moraga. Editorial Rama. Aceptado, pendiente de publicación en 2010.

7.2.5. Conferencias nacionales e iberoamericanas

- Molina, F., Cachero, C., Pardillo, J., Toval, A. Metamodelo y perfil UML para el modelado orientado a metas de requisitos medibles. 13th Conference on Software Engineering and Databases (JISBD'08), pp. 345-356. Gijón, España. 2008.

- Molina, F., Toval, A. Una propuesta para la evaluación temprana de usabilidad en Sistemas de Información Web. IX Congreso Internacional de Interacción Persona-Ordenador (Interaction'08), pp. 243-252. Albacete, España. 2008.
- Molina F., Lucas, F.J., Toval, A., Vara, J.M., Marcos, E.. Soporte CASE para el Desarrollo Preciso de Sistemas de Información Web. Conferencia Ibero-Americana IADIS WWW/Internet 2006, pp. 49-56. Murcia, España. IADIS Press. ISBN: 972-8924-20-8

7.2.6. Proyectos Fin de Carrera

A continuación se muestran los Proyectos Fin de Carrera, en cuya dirección ha participado el doctorando, y que tuvieron como marco líneas de investigación relacionadas con esta tesis doctoral.

- "Generación automática de especificaciones formales en Maude a partir de modelos EMF". Proyecto Fin de Carrera de la titulación Ingeniería en Informática. Autora: María del Mar Ivorra Alfonso. Directores: Francisco J. Lucas, Fernando Molina y Ambrosio Toval. Calificación: Matrícula de Honor. Julio, 2008.
- "Generación y verificación automática de modelos en SIW". Proyecto Fin de Carrera de la titulación Ingeniería en Informática. Autor: J. Miguel Alcaraz. Directores: Fernando Molina, Ambrosio Toval. Calificación: Sobresaliente. Julio, 2007.
- "Un plugin para V&V de Sistemas de Información Web". Proyecto Fin de Carrera de la titulación Ingeniería en Informática. Autoras: Ana Cortés y Josefina Torregrosa. Directores: Fernando Molina, Francisco J. Lucas, Ambrosio Toval. Calificación: Sobresaliente. Septiembre, 2007.

7.2.7. Informes técnicos

- Molina, F., Cachero, C., Pardillo, J., Toval, A. A Systematic Review of Requirements Metamodels. Departamento de Informática y Sistemas, Universidad de Murcia. 2009.

7.3. Líneas de trabajo futuro

Aunque las propuestas mostradas en cada uno de los capítulos anteriores contribuyen al objetivo común de mejorar los artefactos software creados en las fases iniciales de desarrollo de proyectos software, su variedad y la amplitud del método propuesto permiten la extensión de éste en diferentes puntos. Al mismo tiempo, y una vez que se ha mostrado la viabilidad de las contribuciones y se han abierto nuevas líneas de investigación, se sugiere abordar el trabajo futuro enfocándose en cada momento en tan sólo alguna de las líneas de trabajo que se detallan a continuación, trabajando en la misma en profundidad

y haciendo crecer de esta forma la utilidad del método en lugar de extender la anchura del mismo.

Con respecto a las técnicas de metamodelado de requisitos, se debe seguir trabajando en el metamodelo propuesto y en su extensión para considerar nuevos conceptos como características de trazabilidad entre requisitos o análisis de conflictos. El uso de este metamodelo en un ámbito de desarrollo global de software constituirá un campo de aplicación que permitirá obtener un valioso *feedback* para mejorar este metamodelo. El trabajo realizado en esta línea se relaciona con el llevado a cabo en la tesis de Joaquín Nicolás [203], en su propuesta preliminar para un método de Ingeniería de Requisitos para desarrollo global. En relación con el metamodelado, seguiremos trabajando en la integración con otras propuestas de metamodelado de requisitos, dentro del grupo, como el propuesto en la tesis de Begoña Moros [204] ya en su fase final, que trata con mayor énfasis cuestiones importantes como la trazabilidad y reutilización de requisitos y, en el futuro, las técnicas que propone también serán usadas en el ámbito del desarrollo global de software.

Otro área de trabajo relacionada con la anterior consiste en la definición de notaciones visuales que acompañen a los metamodelos desarrollados. En este sentido, y en colaboración con el Grupo de Investigación Lucentia de la Universidad de Alicante, se ha comenzado a trabajar en una serie de experimentos encaminados a encontrar una notación (sintaxis concreta) que facilite a los modeladores la construcción de modelos de requisitos. De nuevo en este punto las particularidades de los entornos globales de desarrollo de software introducen interesantes problemas de investigación. Por ejemplo, las diferencias culturales entre modeladores de diferentes países o entornos pueden provocar que una única notación no sea útil para todos los nodos que contribuyen al desarrollo del sistema. Por ello, se debe trabajar en la definición de notaciones neutras que eviten estos problemas o en diferentes notaciones que reflejen estas particularidades culturales junto con transformaciones automáticas entre éstas que faciliten el intercambio de modelos de requisitos creados por diferentes modeladores.

La dos líneas de trabajo futuro mencionadas hasta el momento se abordarán en el marco del proyecto PANGEA (ver Sección 1.3.2), orientado específicamente al desarrollo de software en entornos globales.

En cuanto a la propuesta ModelSec para la consideración de requisitos de seguridad, consideramos que se trata de una línea de investigación muy extensa que puede seguir ampliándose tratando aspectos como nuevos mecanismos de control de acceso y políticas de seguridad o la generación de código para diferentes lenguajes y plataformas (por ejemplo, diferentes sistemas gestores de bases de datos). El trabajo en la línea de seguridad se complementará con el realizado en la tesis de Joaquín Lasheras [205], en la que se propone un método para la consideración de requisitos de seguridad basado en reutilización y análisis de riesgos.

Con respecto a la propuesta de evaluación temprana de requisitos y métricas de usabilidad, se abren varias líneas de trabajo futuro. La línea más inmediata consiste en la ampliación del catálogo de requisitos y métricas que puedan ser definidos y evaluados sobre modelos navegacionales. Otras dos líneas de trabajo que pueden ser abordadas

están relacionadas con la consideración de otros atributos de calidad, como la accesibilidad, y la aplicación de una propuesta similar sobre otros tipos de modelos, como es el caso de los modelos de presentación.

En relación con la aplicación de técnicas formales para el análisis de modelos conceptuales, una línea de trabajo directa consiste en la ampliación del número de chequeos aplicables sobre modelos. Además, el uso de especificaciones algebraicas también puede ser usado para llevar a cabo actividades de análisis de consistencia entre modelos en el área de Ingeniería Web. En la tesis doctoral de Francisco J. Lucas [206] se propone un enfoque formal para análisis de consistencia entre modelos que puede trasladarse al área del desarrollo web. Esta línea de trabajo ha dado sus primeros resultados iniciales [33] y, en el futuro, deberá ser abordada en mayor profundidad.

Por último, la mejora y evolución de las herramientas de soporte y la validación de las contribuciones en ámbitos reales de desarrollo y proyectos de transferencia tecnológica son dos aspectos que deben seguir abordándose de forma continuada.

Bibliografía

[1] Escalona, M., Aragón, G.: NDT. A Model-Driven Approach for Web Requirements. IEEE Trans. Software Eng. **34(3)** (2008) 377–390

[2] García, F., Bertoa, M.F., Calero, C., Vallecillo, A., Ruiz, F., Piattini, M., Genero, M.: Towards a consistent terminology for software measurement. Information & Software Technology **48(8)** (2006) 631–644

[3] StandishGroup: The Chaos Report. http://www.standishgroup.com/ (2009)

[4] Charette, R.: Why Software Fails. IEEE Spectrum (2005) 36–43

[5] Schmidt, D.C.: Guest Editor's Introduction: Model-Driven Engineering. Computer **39(2)** (2006) 25–31

[6] Selic, B.: The Pragmatics of Model-Driven Development. IEEE Software **20(5)** (2003) 19–25

[7] Goknil, A., Kurtev, I., van den Berg, K.: A Metamodeling Approach for Reasoning about Requirements. In: ECMDA-FA. (2008) 310–325

[8] Lucas, F.J., Molina, F., Toval, A.: A Systematic Review of UML Model Consistency Management. Information and Software Technologies **51(12)** (2009) 1631–1645

[9] Muskens, J., Bril, R.J., Chaudron, M.: Generalizing Consistency Checking between Software Views. In: WICSA. (2005) 169–180

[10] Nuseibeh, B., Easterbrook, S.M., Russo, A.: Making inconsistency respectable in software development. Journal of Systems and Software **58(2)** (2001) 171–180

[11] Selic, B.: Model-Based Software Engineering: Expected and Unexpected Challenges. In: Keynote at 13th Conference on Software Engineering and Databases. (2008)

[12] Ginige, A., Murugesan, S.: Guest Editors Introduction: Web Engineering - An Introduction. IEEE MultiMedia **8(1)** (2001) 14–18

[13] Lowe, D.: Web system requirements: an overview. Req. Eng. Journal **8(2)** (2003) 102–113

[14] Murugesan, S.: Web Application Development: Challenges And The Role Of Web Engineering. In Web Engineering: Modelling and Implementing Web Applications (2007) 7–32

[15] Kappel, G., Michlmayr, E., Pröll, B., Reich, S., Retschitzegger, W.: Web Engineering - Old Wine in New Bottles? In: ICWE. (2004) 6–12

[16] Epner, M.: Poor Project Management Number-One Problem of Outsourced E-Projects. Research Briefs, Cutter Consortium (2000)

[17] Lang, M., Fitzgerald, B.: Web-based systems design: a study of contemporary practices and an explanatory framework based on "method-in-action". Requir. Eng. **12(4)** (2007) 203–220

[18] Moreno, N., Romero, J.R., Vallecillo, A.: An Overview of Model-Driven Web Engineering and the MDA. In: Web Engineering and Web Applications Design Methods. Chapter 12. (2007)

[19] Juristo, N., Moreno, A., Sánchez, M.: Guidelines for Eliciting Usability Functionalities. IEEE Trans. Software Eng. **33(11)** (2007) 744–758

[20] Juristo, N., Moreno, A., Sánchez, M.: Analysing the impact of usability on software design. Journal of Systems and Software **80(9)** (2007) 1506–1516

[21] Glisson, W., McDonald, A., Welland, R.: Web engineering security: a practitioner's perspective. In: ICWE '06: Proceedings of the 6th international conference on Web engineering, New York, NY, USA, ACM (2006) 257–264

[22] Molina, F., Toval, A.: Integrating usability requirements that can be evaluated in design time into Model Driven Engineering of Web Information Systems. Advances in Engineering Software **40(12)** (2009) 1306–1317

[23] Molina, F., Pardillo, J., Toval, A.: Modelling Web-Based Systems Requirements Using WRM. In: WISE Workshops. (2008) 122–131

[24] Pardillo, J., Molina, F., Cachero, C., Toval, A.: A UML Profile for Modelling Measurable Requirements. In: ER Workshops. (2008) 123–132

[25] Molina, F., Pardillo, J., Cachero, C., Toval, A.: An MDE Modelling Framework for Measurable Goal-oriented Requirements. Accepted in International Journal of Intelligent Systems. Special Issue on Goal-Driven Requirements Engineering (2010)

[26] Molina, F., Pardillo, J., Cachero, C., Toval, A.: Metamodelo y perfil UML para el modelado orientado a metas de requisitos medibles. In: Jornadas de Ingeniería del Software y Bases de Datos. (2008)

[27] Molina, F., Pardillo, J., Cachero, C., Toval, A.: Towards a Requirements-Aware Common Web Engineering Metamodel. In: LA-WEB, (IEEE Press) 75–82

[28] Molina, F., Toval, A.: A Generic Approach to Improve Navigational Model Usability Based Upon Requirements and Metrics. In: WISE Workshops. (2007) 511–516

[29] Sanchez, O., Molina, F., Garcia-Molina, J., Toval, A.: A model driven approach for generating code from security requirements. In: WOSIS'09: Int. Workshop on Security In Information Systems. (2009)

[30] Sanchez, O., Molina, F., Garcia-Molina, J., Toval, A.: Modelsec: a generative architecture for model-driven security. Accepted in Special Issue 'Security in Information Systems: New Advances and Tendencies' of the Journal of Universal Computer Science. To appear (2010)

[31] Molina, F., Lucas, F.J., Toval, A., Vara, J.M., Cáceres, P., Marcos, E.: Towards Quality Web Information Systems Through Precise Model Driven Development. In: Handbook of Research on Web Information Systems Quality, Idea Group (2007) 317–355

[32] Lucas, F.J., Molina, F., Toval, A., de Castro, V., Cáceres, P., Marcos, E.: Precise wis development. In: ICWE. (2006) 71–76

[33] Lucas, F., Molina, F., Toval, A.: A Rule-based Approach for Managing Semantic Inconsistencies in Web Information Systems Development. In: 5th Int. Workshop on Automated Specification and Verification of Web Systems. In Press. (2009)

[34] Glass, R.: Software Engineering: Facts and Fallacies. Addison-Wesley (2002)

[35] Glass, R.L.: Software Runaways: Monumental Disasters. Prentice Hall (1998)

[36] Selic, B.: Personal reflections on automation, programming culture, and model-based software engineering. Autom. Softw. Eng. **15(3-4)** (2008) 379–391

[37] Escalona, M.J., Koch, N.: Requirements Engineering for Web Applications: A Comparative Study. Journal of Web Engineering **3** (2004) 193–212

[38] Meier, J.: Web application security engineering. IEEE Security and Privacy **4(4)** (2006) 16–24

[39] Damian, D., Chisan, J.: An Empirical Study of the Complex Relationships between Requirements Engineering Processes and Other Processes that Lead to Payoffs in Productivity, Quality, and Risk Management. IEEE Trans. SW Eng. **32(7)** (2006) 433–453

[40] Nuseibeh, B., Easterbrook, S.: Requirements Engineering: a roadmap. In: ICSE '00: Proceedings of the Conference on The Future of Software Engineering, New York, NY, USA, ACM (2000) 35–46

[41] Brooks, F.: No Silver Bullet Essence and Accidents of Software Engineering. Computer **20(4)** (1987) 10–19

[42] van Lamsweerde, A.: Goal-Oriented Requirements Engineering: From System Objectives to UML Models to Precise Software Specifications. In: ICSE. (2003) 744–745

[43] Rashid, A., Moreira, A., Tekinerdogan, B.: Special issue on Early aspects: aspect-oriented requirements engineering and architecture design. IEE Proceedings - Software **151(4)** (2004) 153–156

[44] Kotonya, G.: Practical Experience with Viewpoint-Oriented Requirements Specification. Requir. Eng. **4(3)** (1999) 115–133

[45] Nuseibeh, B., Kramer, J., Finkelstein, A.: A Framework for Expressing the Relationships Between Multiple Views in Requirements Specification. IEEE Trans. Software Eng. **20(10)** (1994) 760–773

[46] Robertson, S., Robertson, J.: Mastering the requirement process. Addison-Wesley, Boston, MA, USA (1999)

[47] Seidewitz, E.: What models mean. IEEE Software **20(5)** (2003)

[48] Jureta, I., Mylopoulos, J., Faulkner, S.: Revisiting the core ontology and problem in requirements engineering. In: RE. (2008) 71–80

[49] Glinz, M.: A Risk-Based, Value-Oriented Approach to Quality Requirements. IEEE Software **25(2)** (2008) 34–41

[50] Fenton, N.E., Pfleeger, S.L.: Software Metrics: A Rigorous and Practical Approach. PWS Publishing Co., Boston, MA, USA (1998)

[51] Nuseibeh, B., Robertson, S.: Making requirements measurable. In: RE. (1997) 262

[52] Boehm, B.W., Brown, J.R., Lipow, M.: Quantitative evaluation of software quality. In: ICSE '76: 2nd Int. Conf. on Software Engineering, Los Alamitos, CA, USA, IEEE Computer Society Press (1976) 592–605

[53] García, F., Ruiz, F., Calero, C., Bertoa, M., Vallecillo, A., Mora, B., Piattini, M.: Effective use of ontologies in software measurement. Knowl. Eng. Rev. **24(1)** (2009) 23–40

[54] I.S.O.: ISO/IEC 9126-1. Softw. Eng.-Product Quality - Part 1: Quality Model. (2000)

[55] MDWEnet-Initiative: Web Engineering Metamodel. http://www.pst.ifi.lmu.de/projekte/mdwenet/index (2008)

[56] Moreno, N., Vallecillo, A.: Towards interoperable Web Engineering methods. Journal of the American Society for Information Science and Technology **59(7)** (2008) 1073–1092

[57] OMG: MDA Guide Version 1.0.1, http://www.omg.org/mda. (2003)

[58] OMG: Object management group. (http://www.omg.org/)

[59] Kent, S.: Model Driven Engineering. In: IFM. (2002) 286–298

[60] France, R., Bieman, J.: Multi-View Software Evolution: A UML-based Framework for Evolving Object-Oriented Software. In: IEEE Int. Conf. on Software Maintenance, IEEE Computer Society (2001) 386

[61] OMG: Meta Object Facility Core Specification version 2.0. http://www.omg.org/spec/MOF/2.0/ (2006)

[62] OMG: MOF QVT Final Adopted Specification. Object Management Group., Retrieved from: http://www.omg.org/docs/ptc/07-07-07.pdf. (2007)

[63] Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I., Valduriez, P.: ATL: a QVT-like transformation language. 21th ACM SIGPLAN, OOPSLA, Oregon, USA (2006)

[64] Cuadrado, J.S., Molina, J.G.: Approaches for Model Transformation Reuse: Factorization and Composition. In: ICMT. (2008) 168–182

[65] Cuadrado, J.S., Molina, J.G., Tortosa, M.M.: RubyTL: A Practical, Extensible Transformation Language. In: ECMDA-FA. (2006) 158–172

[66] SmartQVT: SmartQVT. http://smartqvt.elibel.tm.fr/ (2007)

[67] IKV++: Medini QVT - IKV++ technologies. http://projects.ikv.de/qvt. Último acceso: Octubre 2009 (2008)

[68] IBM: Rational Software Design. http://www.ibm.com/software/rational (2008)

[69] AndroMDA: AndroMDA. http://www.andromda.org/ (2006)

[70] Boronat, A., Carsí, J.A., Ramos, I.: Algebraic Specification of a Model Transformation Engine. In: FASE. (2006) 262–277

[71] Boronat, A., Carsí, J.A., Ramos, I.: Automatic Support for Traceability in a Generic Model Management Framework. In: ECMDA-FA. (2005) 316–330

[72] Greenfield, J., Short, K., Cook, S., Kent, S.: Software Factories: Assembling Applications with Patterns, Models, Frameworks and Tools. John Wiley&Sons (2004)

[73] Eclipse: Eclipse Modeling Framework Project (EMF). http://www.eclipse.org/modeling/emf/ (2007)

[74] OMG: MDA Success Stories. http://www.omg.org/mda/productssuccess.htm (2008)

[75] Hossler, J., Born, M., Saito, S.: Significant Productivity Enhancement through Model Driven Techniques: A Success Story. In: EDOC'06: 10th IEEE International Enterprise Distributed Object Computing Conference, Washington, DC, USA, IEEE Computer Society (2006) 367–373

[76] Selic, B.: Wherefore art thou, model-driven development? In: Keynote at ECMDA-FA. (2006)

[77] Haley, C., Laney, R., Moffett, J., Nuseibeh, B.: Security Requirements Engineering: A Framework for Representation and Analysis. IEEE Trans. Software Eng. **34(1)** (2008) 133–153

[78] Fernández-Medina, E., Jurjens, J., Trujillo, J., Jajodia, S.: Editorial: Model-driven development for secure information systems. Inf. Softw. Technol. **51(5)** (2009) 809–814

[79] Mouratidis, H., Giorgini, P.: Integrating Security and Software Engineering: Advances and Future Visions. Idea Group (2006)

[80] van Lamsweerde, A.: Elaborating security requirements by construction of intentional anti-models. In: ICSE '04: Proceedings of the 26th International Conference on Software Engineering, Washington, DC, USA, IEEE Computer Society (2004) 148–157

[81] Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: Tropos: Agent-Oriented Software Development Methodology. Journal of Autonomous Agents and Multi-Agent System **8(3)** (2004) 203–236

[82] Bresciani, P., Mouratidis, H., Zanone, N.: Modelling Security and Trust with Secure Tropos. Integrating Security and Software Engineering: Advances and Future Visions (2007) 160–189

[83] Yu, E., Liu, L., Mylopoulos, J.: A Social Ontology for Integrating Security and Software Engineering. Integrating Security and Software Engineering: Advances and Future Visions (2007) 70–109

[84] Firesmith, D.: Engineering security requirements. Journal of Object Technology **2** (2003) 53–68

[85] Basin, D., Doser, J., Lodderstedt, T.: Model driven security: From UML models to access control infrastructures. ACM Trans. Softw. Eng. Methodol. **15(1)** (2006) 39–91

[86] OMG: Unified Modeling Language (UML), version 2.1.1. http://www.omg.org/technology/documents/formal/uml.htm (2007)

[87] Jurjens, J.: Secure Systems Development with UML. Springer Verlag (2003)

[88] Jurjens, J., Schreck, J., Bartmann, P.: Model-based security analysis for mobile communications. In: ICSE. (2008) 683–692

[89] Kelly, S., Tolvanen, J.: Domain-Specific Modeling: Enabling Full Code Generation. Wiley-IEEE Computer Society Press (2008)

[90] Voelter, M.: MD* Best Practices. http://www.voelter.de/data/articles/DSLBestPractices-Website.pdf. (2008)

[91] Lang, U., Schreiner, R.: Managing business compliance using model-driven security management. In: Securing Electronic Business Processes. (2008) 1–11

[92] Garzotto, F., Paolini, P., Schwabe, D.: HDM—a model-based approach to hypertext application design. ACM Trans. Inf. Syst. **11(1)** (1993) 1–26

[93] Isakowitz, T., Stohr, E.A., Balasubramanian, P.: RMM: a methodology for structured hypermedia design. Commun. ACM **38(8)** (1995) 34–44

[94] Isakowitz, T., Kamis, A., Koufaris, M.: The Extended RMM Methodology for Web Publishing. Information Systems Working Papers Series (1998)

[95] Schwabe, D., Rossi, G.: The object-oriented hypermedia design model. Commun. ACM **38(8)** (1995) 45–46

[96] Schwabe, D., Rossi, G.: An object oriented approach to web-based applications design. Theor. Pract. Object Syst. **4(4)** (1998) 207–225

[97] Baresi, L., Garzotto, F., Paolini, P.: Extending UML for Modeling Web Applications. In: HICSS '01:34th Annual Hawaii International Conference on System Sciences - Vol. 3, Washington, DC, USA, IEEE Computer Society (2001) 3055

[98] Hennicker, R., Koch, N.: A UML-based Methodology for Hypermedia Design. In: Proc. of UML 2000 Conference, Springer Verlag (2001) 410–424

[99] Díaz, P., Aedo, I., Montero, S.: Ariadne, a development method for hypermedia. In: DEXA'01: 12th Int. Conf. on Database and Expert Systems Applications, London, UK, Springer-Verlag (2001) 764–774

[100] Mecca, G., Atzeni, P., Masci, A., Merialdo, P., Sindoni, G.: The Araneus Web-Base Management System. In: SIGMOD Conference. (1998) 544–546

[101] Atzeni, P., Merialdo, P., Mecca, G.: Data-Intensive Web Sites: Design and Maintenance. World Wide Web **4(1-2)** (2001) 21–47

[102] Ceri, S., Fraternali, P., Bongio, A.: Web Modeling Language (WebML): a Modeling Language for Designing Web Sites. WWW9 Conference (2000)

[103] De Troyer, O.M.F., Leune, C.J.: WSDM: a user centered design method for Web sites. Comput. Netw. ISDN Syst. **30(1-7)** (1998) 85–94

[104] Fraternali, P., Paolini, P.: Model-driven development of Web applications: the AutoWeb system. ACM Trans. Inf. Syst. **18(4)** (2000) 323–382

[105] Conallen, J.: Building Web applications with UML. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2000)

[106] Cachero, C.: OO-H: una extensión a los métodos OO para el modelado y generación automática de interfaces hipermediales. PhD thesis, Universidad de Alicante (2003)

[107] Gómez, J., Cachero, C., Pastor, O.: Conceptual Modeling of Device-Independent Web Applications. IEEE MultiMedia **8(2)** (2001) 26–39

[108] Cáceres, P., Marcos, E., vela, B.: A MDA-Based Approach for Web Information System Development. In: Software Model Engineering Workshop (WiSME), Available in: http://www.metamodel.com/wisme-2003/ (2003)

[109] Acerbis, R., Bongio, A., Brambilla, M., Butti, S.: WebRatio 5: An Eclipse-Based CASE Tool for Engineering Web Applications. In: ICWE. (2007) 501–505

[110] Acerbis, R., Bongio, A., Butti, S., Ceri, S., Ciapessoni, F., Conserva, C., Fraternali, P., Toffetti, G.: Webratio, an innovative technology for web application development. In: ICWE. (2004) 613–614

[111] Knapp, A., Koch, N., Zhang, G., Hassler, H.: Modeling business processes in web applications with argouwe. In: UML. (2004) 69–83

[112] Vara, J.: M2DAT: a Technical Solution for Model-Driven Development of Web Information Systems. PhD thesis, Universidad Rey Juan Carlos (Madrid) (2009)

[113] de Castro, V.: Aproximación MDA para el desarrollo orientado a servicios de Sistemas de Información Web: del modelo de negocio al modelo de composición de servicios web. PhD thesis, Universidad Rey Juan Carlos (Madrid) (2007)

[114] Matera, M., Rizzo, F., Cortázar, R., Perallos, A.: The Usability Dimension in the Development of Web Applications. In: Handbook of Research on Web Information Systems Quality, Idea Group Publishing (2008) 235–249

[115] Retschitzegger, W., Schwinger, W.: Towards Modeling of DataWeb Applications - A Requirements' Perspective. In: Proc. of the Americas Conference on Information Systems (AMCIS). Volume 1. (2000) 149–155

[116] Insfran, E., Pastor, O., Wieringa, P.: Requirements engineering-based conceptual modelling. Requirements Engineering Journal **7(2)** (2002) 61–72

[117] Vilain, P., Schwabe, D., de Souza, C.S.: A Diagrammatic Tool for Representing User Interaction in UML. In: UML. (2000) 133–147

[118] Bolchini, D., Paolini, P.: Goal-driven requirements analysis for hypermedia-intensive Web applications. Requirement Engineering Journal **9(2)** (2004) 85–103

[119] Valderas, P., Pelechano, V.: Introducing requirements traceability support in model-driven development of web applications. Information and Software Technology **51(4)** (2009) 749–768

[120] Cuaresma, M.J.E., Koch, N.: Metamodeling the Requirements of Web Systems. In: WEBIST (Selected Papers). (2006) 267–280

[121] Yu, E.S.K.: Towards modelling and reasoning support for early-phase requirements engineering. In: RE, IEEE Computer Society. (1997) 226–235

[122] Toval, A., Moros, B., Nicolás, J., Lasheras, J.: Eight Key Issues for an Effective Reuse. Int. Journal of Computer Systems Science **23(5)** (2008)

[123] Nielsen, J.: Designing Web Usability: The practice of Simplicity. New Riders Publishing (2000)

[124] I.S.O.: ISO 9241-11. Ergonomic Requirements for office work with visual display terminals (VDT)s - Part 11: Guidance on usability. (1998)

[125] Donahue, G.M.: Usability and the Bottom Line. IEEE Software **18(1)** (2001)

[126] Nielsen, J.: Return on Investment for Usability. Alertbox (2003)

[127] Wroblewski, L.: Web Form Design: Filling in the Blanks. Rosenfeld Media (2008)

[128] (W3C), W.W.W.C.: Web Accessibility Initiative (WAI). http://www.w3.org/WAI/ (2008)

[129] Ruiz, J., Calero, C., Piattini, M.: Classifying Web Metrics. In: Software Audit and Metrics. (2004) 22–37

[130] Dhyani, D., NG, W., Bhowmick, S.: A Survey of Web Metrics. ACM Computing Surveys, 34(4) (2002) 469–503

[131] Seffah, A., Metzker, E.: The obstacles and myths of usability and software engineering. Commun. ACM **47(12)** (2004) 71–76

[132] Ivory, M.Y., Hearts, M.: An Empirical Foundation for Automated Web Interface Evaluation. PhD thesis, University of California at Berkeley (2001)

[133] Abascal, J., Arrue, M., Garay, N., Tomás, J.: EvalIris: A Web Service for Web Accessibility Evaluation. 12th Int. WWW Conference, Budapest, Hungary (2003)

[134] T.A.W.: Test de Accesibilidad Web. http://www.tawdis.net (2009)

[135] W3C: The W3C Validation Service. http://validator.w3.org/ (2009)

[136] Abrahao, S., Insfran, E.: Early usability evaluation in Model-Driven Architecture Environments. In: 6th IEEE Int. Conference on Quality Software, IEEE Press, Wiley. (2006)

[137] Atterer, R., Schmidt, A.: Extending Web Engineering Models and Tools For an Automatic Usability Validation. Journal of Web Engineering **1** (2006) 43–64

[138] Binder, R.: Testing object-oriented systems: models, patterns and tools. Addison-Wesley Object Technology Series (1999)

[139] Alpuente, M., Ballis, D., Falaschi, M.: Rule-based verification of web sites. STTT **8(6)** (2006) 565–585

[140] Ehrig, H., Mahr, B.: Fundamentals of Algebraic Specification. Equations and Initial Semantic. ISBN 3-540-13718-1 Springer-Verlag. 1985 (1985)

[141] Alpuente, M., Ballis, D., Falaschi, M., Romero, D.: A Semi-Automatic Methodology for Repairing FaultyWeb Sites. Int. Conf. on Software Engineering and Formal Methods (2006) 31–40

[142] Ballis, D., García-Vivó, J.: A Rule-based System for Web site Verification. Electr. Notes Theor. Comput. Sci. **157(2)** (2006) 11–17

[143] Ballis, D., García-Vivó, J.: A rewriting-based system for web site verification. In: WWV. (2005) 153–156

[144] Alpuente, M., Ojeda, P., Romero, D., Ballis, D., Falaschi, M.: An Abstract Generic Framework for Web Site Verification. In: SAINT. (2008) 104–110

[145] Flores, S., Lucas, S., Villanueva, A.: Formal Verification of Websites. Electr. Notes Theor. Comput. Sci. **200(3)** (2008) 103–118

[146] Brambilla, M., Cabot, J., Moreno, N.: Tool Support for Model Checking of Web Application Designs. In: ICWE. (2007) 533–538

[147] Haidar, A.N., Abdallah, A.E.: Towards a Formal Framework for Developing Secure Web Services. In: WWV'06: 2nd Int. Workshop on Automated Specification and Verification of Web Systems, Washington, DC, USA, IEEE Computer Society (2006) 61–70

[148] Beckert, B., Hoare, T., Hähnle, R., Smith, D.R., Green, C., Ranise, S., Tinelli, C., Ball, T., Rajamani, S.K.: Intelligent Systems and Formal Methods in Software Engineering. IEEE Intelligent Systems **21(6)** (2006) 71–81

[149] Foster, H., Mayer, P.: Leveraging Integrated Tools for Model-Based Analysis of Service Compositions. In: Int. Conf. on Internet and Web Applications and Services (ICIW 2008), IEEE Computer Society (2008) 72–77

[150] Mayer, P., Ráth, I., Horváth, A.: Report on the Sensoria Development Environment (second version). Project deliverable D7.4.c, SENSORIA EU-IST Project (2008)

[151] Muskens, J., Bril, R., Chaudron, M.: Generalizing Consistency Checking between Software Views. 5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05) (2005) 169–180

[152] Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.L., eds.: All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic. In Clavel, M., Durán, F.,

Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.L., eds.: All About Maude. Volume 4350 of LNCS., Springer (2007)

[153] Eclipse: Eclipse graphical modeling framework. http://www.eclipse.org/gmf/ (2008)

[154] Eclipse: Generative Modeling Technologies (GMT): MOFScript. http://www.eclipse.org/gmt/ (2008)

[155] OMG: Software Process Engineering Metamodel Specification. (2007)

[156] Kleppe, A., Warmer, J., Bast, W.: MDA Explained: The Model Driven Architecture: Practice and Promise. Addison-Wesley, Boston, MA, USA (2003)

[157] Vicente-Chicote, C., Moros, B., Toval, A.: REMM-Studio: an Integrated Model-Driven Environment for Requirements Specification, Validation and Formatting. Journal of Object Technology, Special Issue TOOLS EUROPE 2007 **6(9)** (2007) 437–454

[158] Koch, N., Zhang, G., Cuaresma, M.J.E.: Model transformations from requirements to web system design. In: ICWE. (2006) 281–288

[159] Martínez, A., Pastor, O., Mylopoulos, J., Giorgini, P.: From Early to Late Requirements: A Goal-Based Approach. In: AOIS. (2006) 123–142

[160] Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: Tropos: An agent-oriented software development methodology. Autonomous Agents and Multi-Agent Systems **8(3)** (2004) 203–236

[161] Dardenne, A., van Lamsweerde, A., Fickas, S.: Goal-directed requirements acquisition. Science of Computer Programming **20(1-2)** (1993) 3–50

[162] Mylopoulos, J., Yu, E., Nixon, B.A., Chung, L.: Non-Functional Requirements in Software Engineering. The Kluwer International Series in Software Engineering (2000)

[163] Kitchenham, B.: Guidelines for performing systematic literature reviews in software engineering. EBSE Technical Report EBSE-2007-01, Software Engineering Group, School of Computer Science and Mathematics, Keele University, UK and Departament of Computer Science, University of Durham, Durham, UK (2007)

[164] Molina, F., Pardillo, J., Cachero, C., Toval, A.: A Systematic Review of Requirements Metamodels. Technical report, University of Murcia. Available on http://www.lucentia.es/index.php/MRM (2008)

[165] López, O., Laguna, M.A., García, F.J.: Metamodeling for Requirements Reuse. In: WER. (2002) 76–90

[166] Berre, A.J.: COMET (Component and Model based Development Methodology). http://modelbased.net/comet/ (2006)

[167] Fernández, J.L., Monzón, A.: A Metamodel and a Tool for Software Requirements Management. In: Reliable Software Technologies. Ada-Europe. Berlin (Germany). (2000)

[168] Vogel, R., Mantell, K.: MDA adoption for a SME: evolution, not revolution - Phase II. In: 2nd Workshop "From code centric to model centric software engineering: Practices, Implications and ROI. In conjuntion with ECMDA 2006. (2006)

[169] Toval, A., Nicolás, J., Moros, B., García, F.: Requirements Reuse for Improving Information System Security: A practicioner's Approach. Req. Eng. Journal **6(4)** (2002) 205–219

[170] IEEE: Std 830-1998: Guide to Software Requirements Specifications (ANSI). In Volume 4: Resource and Technique Standards. The Institute of Electrical and Electronics Engineers, Inc. IEEE Software Engineering Standards Collection (1998)

[171] Larman, C.: Applying UML and Patterns. (2005)

[172] Serrano, M.A., Calero, C., Sahraoui, H.A., Piattini, M.: Empirical studies to assess the understandability of data warehouse schemas using structural metrics. Software Quality Journal **16(1)** (2008) 79–106

[173] Blaine, J.D., Cleland-Huang, J.: Software quality requirements: How to balance competing priorities. IEEE Softw. **25(2)** (2008) 22–24

[174] Larman, C.: Applying UML and Patterns. (2004)

[175] Cachero, C., Melia, S., Genero, M., Poels, G., Calero, C.: Towards improving the navigability of web applications: a model-driven approach. European Journal of Information Systems **16** (2007) 420–447

[176] Briand, L., Morasca, S., Basili, V.: Defining and validating measures for object-based high-level design. IEEE Trans. Softw. Eng. **25(5)** (1999) 722–743

[177] Briand, L., Morasca, S., Basili, V.: Property-based software engineering measurement. IEEE Trans. on Soft. Eng. **22** (1996) 68–86

[178] Moody, D.: What Makes a Good Diagram? Improving the Cognitive Effectiveness of Diagrams in IS Development. In: Advances in Information Systems Development, Springer (2006) 481–492

[179] Whitley, K.N.: Visual Programming Languages and the Empirical Evidence For and Against. J. Vis. Lang. Comput. **8(1)** (1997) 109–142

[180] Green, T., Petre, M.: Usability Analysis of Visual Programming Environments: A Cognitive Dimensions Framework. Journal of Visual Languages and Computing **7(2)** (1996) 131–174

[181] Ayala, C.P., Cares, C., Carvallo, J.P., Grau, G., Haya, M., Salazar, G., Franch, X., Mayol, E., Quer, C.: A comparative analysis of i*-based agent-oriented modelling languages. In: Software Engineering and Knowledge Engineering (SEKE'05). (2005) 43–50

[182] Amyot, D., Mussbacher, G.: Urn: Towards a new standard for the visual description of requirements. In: SAM. (2002) 21–37

[183] Moody, D.: Comparative Evaluation of Large Data Model Representation Methods: The Analyst's Perspective. In: ER. (2002) 214–231

[184] Mazón, J.N., Pardillo, J., Trujillo, J.: A Model-Driven Goal-Oriented Requirement Engineering Approach for Data Warehouses. In: ER Workshops. (2007) 255–264

[185] Object Management Group: Object Constraint Language (OCL), version 2.0. http://www.omg.org/technology/documents/formal/ocl.htm (2006)

[186] Larkin, J.H., Simon, H.A.: Why a diagram is (sometimes) worth ten thousand words. Cognitive Science **11(1)** (1987) 65–100

[187] Chisholm, W., Vanderheiden, G., Jacobs, I.: Web Content Accessibility Guidelines. Alertbox (2008)

[188] I.S.O.: ISO/IEC 15408 (Common Criteria v3.0): Information Technology Security Techniques-Evaluation Criteria for IT Security. (2005)

[189] Rodríguez, A., Fernández-Medina, E., Piattini, M.: A BPMN Extension for the Modeling of Security Requirements in Business Processes. IEICE Transactions **90-D(4)** (2007) 745–752

[190] Samarati, P., Capitani, S.D.: Access control: Policies, models, and mechanisms. In: FOSAD. (2000) 137–196

[191] Mellado, D., Fernández-Medina, E., Piattini, M.: A common criteria based security requirements engineering process for the development of secure information systems. Comput. Stand. Interfaces **29(2)** (2007) 244–253

[192] Fernández-Medina, E., Piattini, M.: Designing secure databases. Information & Software Technology **47(7)** (2005) 463–477

[193] Cuadrado, J., Molina, J., Tortosa, M.: Rubytl: A practical, extensible transformation language. (2006) 158–172

[194] ORACLE: Oracle label security. http://www.oracle.com/technology/deploy/security/database-security/label-security/index.html (2009)

[195] Escalona, M.J., Torres, J., Mejías, M., Gutiérrez, J.J., Villadiego, D.: The treatment of navigation in Web Engineering. Advances in Engineering Software **38(4)** (2007) 267–282

[196] Abrahão S., Condori N., O.L., O., P.: Defining and Validating Metrics for Navigational Models. METRICS 2003, Sydney, Australia, IEEE Press (2003) 200–210

[197] Bézivin, J.: On the unification power of models. Software and System Modeling 4(2) (2005) 171–188

[198] Eclipse-OCL: Model Development Tools (MDT). http://www.eclipse.org/modeling/mdt/?project=ocl (2008)

[199] Fernández, J.L., Toval, A.: Can Intuition Become Rigorous? Foundations for UML Model Verification Tools. In: ISSRE. (2000) 344–355

[200] Fernández, J.L., Toval, A.: Improving System Reliability via Rigorous Software Modeling: The UML Case. In: IEEE Aerospace Conference. IEEE Computer Society. (2001)

[201] Kurtev, I., Bézivin, J., Aksit, M.: Technological Spaces: An Initial Appraisal. Int. Federated Conf. (DOA, ODBASE,CoopIS), Industrial track, Irvine (2002)

[202] DSIC: Maude Development Tools. http://moment.dsic.upv.es/ (2008)

[203] Nicolás, J.: Una Propuesta de Gestión Integrada de Modelos y Requisitos en Líneas de Productos Software. PhD thesis, Universidad de Murcia (2009)

[204] Moros, B.: Un Proceso de Ingeniería de Requisitos Dirigido por Modelos Centrado en Reutilización. PhD thesis, Universidad de Murcia. En proceso de redacción (2010)

[205] Lasheras, J.: Método de Ingeniería de Requisitos de Seguridad basado en Reutilización y Análisis de Riesgos. PhD thesis, Universidad de Murcia. En proceso de redacción (2010)

[206] Lucas, F.J.: Transformaciones de Modelos Precisas para la Gestión de la Consistencia De Modelos. . PhD thesis, Universidad de Murcia. En proceso de redacción (2010)

Acrónimos

CAWE Computer-Aided Web Engineering

DAC Discretionary Access Control

DSL Domain Specific Languages

EMF Eclipse Modelling Framework

GMF Graphical Modelling Framework

GRL Goal-oriented Requirements Language

HRBAC Hierarchical Role-Based Access Control

IR Ingeniería de Requisitos

MAC Mandatory Access Control

MDA Model Driven Architecture

MDD Model Driven Development

MDE Model Driven Engineering

MDS Model Driven Security

MDWE Model Driven Web Engineering

MOF Meta Object Facility

MRM Measurable Requirements Metamodel

OCL Object Constraint Language

OMG Object Management Group

RMExt Web Engineering Requirements Metamodel Extension

SIW Sistemas de Información Web

SMM Software Measurement Metamodel

SMO Software Measurement Ontology

SPEM Software Process Engineering Metamodel

UML Unified Modeling Language

URM Unified Requirements Metamodel

URN User Requirements Notation

W3C World Wide Web Consortium

WAI Web Accessibility Initiative

WE Web Engineering

Publicaciones

Relevancia

La normativa aplicable para la presentación de esta tesis en el formato de compendio de publicaciones obligaba a la presentación de al menos tres artículos publicados o aceptados para su publicación en revistas indexadas en bases de datos internacionales de reconocido prestigio. Este es el caso de las revistas indexadas con índice de impacto en el listado ISI/JCR. El doctorando presentó tres publicaciones [22, 25, 30] incluidas en este listado, habiéndose desarrollado también un cuarto artículo complementario [8] durante la realización de la tesis doctoral que también aparece en dicho listado y que, igualmente, se incluye en la siguiente relación.

Las revistas *Advances in Engineering Software* e *Information and Software Technology* son revistas bien conocidas que llevan publicándose varias décadas y que cuentan con reconocido prestigio entre la comunidad de Ingeniería del Software, apareciendo ambas en el segundo cuartil (Q2) del listado ISI/JCR (concretamente en las posiciones 39/86 y 37/86 de la categoría *Computer Science* con índices de impacto 1.188 y 1.2, respectivamente, en el listado correspondiente al año 2008). La *International Journal of Intelligent Systems* comenzó a publicarse en el año 1986, cuenta con un factor de impacto de 0.860 y ocupa la posición 69/94 (Q3) en la categoría *Computer Science, Artificial Intelligence.* Por último, el *Journal of Universal Computer Science* lleva publicándose desde 1994 y, entre las publicaciones aportadas, es la que cuenta con índice de impacto más bajo, concretamente de 0.488, ocupando la posición 74/86 en la categoría *Computer Science, Software Engineering.*

A continuación, siguiendo lo indicado en la normativa, se muestran las cartas de aceptación de los artículos aceptados que se encuentran en proceso de publicación y el texto completo de los cuatro artículos mencionados.

Carta de aceptación *International Journal of Intelligent Systems*

Date: 12th October, 2009

Dear Fernando,

Sorry for the delay. We just receive all the review reports back. The decision on your submission "An MDE Modelling Framework for Measurable Goal-oriented Requirements" has been made based on the review comments:

Accept as a full paper

Attached please find [...]

Please, use the attached files to prepare your paper as 'camera ready' and return your final manuscript by November 8.

Carta de aceptación en *Journal of Universal Computer Science*

Date: 1st October, 2009

Dear Fernando,

Thank you for your submission to the Journal of Universal Computer Science.

I am very pleased to inform you that your paper Manuscript entitled "ModelSec: A Generative Architecture for Model-Driven Security" is now acceptable for publication in the special issue entitled "Security in Information Systems: New Advances and Tendencies" of the Journal of Universal Computer Science.

Artículo en *Advances in Engineering Software*

Advances in Engineering Software 40 (2009) 1306–1317

Contents lists available at ScienceDirect

Advances in Engineering Software

journal homepage: www.elsevier.com/locate/advengsoft

Integrating usability requirements that can be evaluated in design time into Model Driven Engineering of Web Information Systems ☆

Fernando Molina *, Ambrosio Toval

Software Engineering Research Group, Department of Informatics and Systems, University of Murcia, Facultad de Informática, Campus de Espinardo, 30100 Murcia, Spain

ARTICLE INFO

Article history:
Received 22 September 2008
Received in revised form 19 November 2008
Accepted 19 January 2009
Available online 10 April 2009

Keywords:
Web engineering
Requirements metamodelling
Usability evaluation
Software quality

ABSTRACT

In recent years, Web Engineering development projects have grown increasingly complex for and critical to the smooth running of organizations. However, recent studies reveal that a high percentage of these projects fail to attain the quality parameters required by stakeholders. The inadequate consideration of requirements management activities together with the absence of attention to the elicitation and evaluation of requirements and metrics related to certain quality attributes which are of special importance in this kind of systems, such as usability, have proved to be some of the main causes of this failure. This paper attempts to reduce some of the quality failures detected in Web Engineering development projects by proposing the consideration and evaluation of quality attributes from early stages of the development process. The presented approach therefore commences with a reinforcement of the requirements related activities in this discipline, which is carried out by using a requirements metamodel. Once these requirements have been identified, the approach focuses on the extension of the conceptual models used by Web Engineering methodologies with the aim of allowing the explicit consideration of usability requirements along with the evaluation of quality metrics during the design of the system. An example of an application illustrating how the approach can be used, along with the automatic support which was developed for it, are also shown.

1. Introduction

The development of Web Information Systems (WIS) has undergone an exponential growth in the last decade. Initially, these systems were used only as a means to disseminate information. However, their complexity has recently increased, they are present in numerous domains being used by millions of users around the world and they have become critical systems for the business strategies of many organizations [36]. This growth, together with some particular features that make WIS development projects different from other software developments [36], have compelled the organizations to adapt their software development processes to deal with the idiosyncrasy of WIS [35]. Moreover, as a result of its awareness of such difficulties, the WIS research community has developed numerous techniques, tools and methodologies within the scope of a new discipline, called Web Engineering (WE), which promotes the establishment and use of sound scientific, engineering and management principles, and disciplined and systematic approaches for the development, deployment and maintenance of web-based systems [19].

However, as various surveys and research publications reveal [31,15], the development of this kind of systems is not exempt from errors and the WIS finally developed, regardless of their scope (business, engineering, scientific, etc.), do not always satisfy the quality requirements demanded by their users. These studies highlight that the top five problem areas of large-scale WE projects are (1) failure to meet business needs (84%), (2) project schedule delays (79%), (3) budget overrun (63%), (4) lack of required functionality (53%) and (5) poor quality of deliverables (52%). In fact these problems, far from being new, are quite similar to those encountered in traditional Software Engineering, in which it has already been proved [9,24] that they are often a symptom of an inadequate management of the tasks related to the requirements workflow of the project.

It is therefore surprising that, in spite of this reality, organizations which carry out the WIS projects or WE methodologies currently used in industry (such as e.g. WebML [8], UWE [33] or OO-H [21]), are still not, to the best of our knowledge, paying the necessary amount of attention to requirements management activities in WIS projects. In fact, various surveys [16,17] reveal that WE methodologies are still mostly focused on the design workflow of web applications, while the requirements workflow is, at best, only tangentially tackled. Another survey carried out by Lang and Fitzgerald [35] over 160 organizations that develop web-based

☆ Partially supported by the projects DEDALO (TIN2006-15175-C05-03) and the CALIPSO thematic network (TIN2005-24055-E), from the Spanish Ministry of Science and Technology, and MELISA (PAC08-0142-3315) The first author is partially funded by the Fundación Séneca (Región de Murcia).

* Corresponding author. Tel.: +34 868 884642/03; fax: +34 868 88 4151.
E-mail addresses: fmolina@um.es (F. Molina), atoval@um.es (A. Toval).

0965-9978/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.advengsoft.2009.01.018

software reveals that 60% of the organizations still consider that one of the main problems in their projects is related to the clarity and stability of requirements.

Furthermore, several authors [36,30] claim that WIS systems require special care when dealing with certain quality attributes (namely usability and accessibility), and tailored usability/accessibility evaluation methods have been proposed for this purpose [25]. Again, organizations and methodologies are ignoring this fact, and continue to rely on informal, ad-hoc quality evaluation and testing activities.

As Blaine and Cleland-Huang [6] notes, the elicitation of quality requirements in current web projects is still mostly implicitly understood by the stakeholders. This fact, together with a validation of this kind of requirements which is more difficult to achieve than that of functional ones [6], is likely to lead to problems with their satisfaction on the delivered products. An analysis of the current WIS development methodologies corroborates these arguments, since they do not offer support in dealing with usability requirements during the development process and they delay this task until the system has been completely developed. The definition of methods for ensuring the usability of web applications is, therefore, one of the current research goals in Web Engineering and Human–Computer Interaction (HCI) [37], and studies such as [30] claim that the consideration and evaluation of usability features should be moved to the early phases of the WIS development lifecycle in order to improve the user satisfaction and decrease maintenance costs [30,3].

In view of this situation, this paper proposes a shift in the way that web development is tackled. This shift is oriented towards a reinforcement of the activities related to the consideration and evaluation of quality attributes such as usability from the early stages of the development process. We therefore provide a requirements metamodel with which to formally define the typical elements that participate in WIS requirements elicitation. This metamodel complements the other metamodels used by WIS methodologies due to their progressive alignment with the model driven approaches [49], emphasizing the importance of requirements elicitation in the quality of software products. Bearing in mind that stakeholders propose certain usability requirements that cannot be expressed over the models used by WE methodologies, the existing WE metamodels have been extended with new entities, attributes and relationships in order to foster the explicit definition and evaluation of usability requirements and metrics during the design of the WIS. The aim of this quality evaluation in design time is to improve models used for WIS development which will later influence the quality of the WIS built from them. The approach focuses on navigational models and is accompanied by a tool that offers automatic support for all the activities of which it is made up.

The integration of these contributions to WIS development processes and methodologies may help to reduce the number of failures detected in WIS development projects. It also provides the benefits related to an adequate requirements management and an early consideration of usability, with the final aim of increasing the quality of the WIS developed and, thus, the stakeholders' satisfaction.

The remainder of the paper is organized as follows. Section 2 presents the approach with which to reinforce requirements related activities in WIS projects. In Section 3, the extension of the navigational models used for WIS development which is carried out to support the definition and evaluation of usability requirements and metrics in early phases of software lifecycle is shown. Section 4 shows the automatic support developed for the approach and, in Section 5, an example that further illustrates the use of the approach is presented. Finally, in Section 6, the main conclusions are drawn and further lines of research are outlined.

2. Reinforcing requirements management activities in WE through a requirements metamodel

The tasks related to requirements are among those which are most influential in the success of a software development project [9]. This is no different in WIS development projects, and the organizations dedicated to their development corroborate this. For example, a set of interviews with organizations that develop web-based systems reported in [36] indicated that, for 76% of the respondents, gathering the right requirements was a critical factor for the success of their projects.

However, requirements management activities remain an open issue in the WE discipline [16] and, moreover, WE methodologies usually ignore that the consideration of quality attributes, such as usability, should start in early stages of the development lifecycle as requirements elicitation and WIS design [29]. The following subsections present an approach which, by using a requirements metamodel as basis, attempts to contribute towards filling this gap, emphasizing the role that RE plays in WE, helping stakeholders in their systematic identification of requirements and paving the way for the consideration of usability requirements from the first phases of WIS development process.

2.1. Requirements metamodelling

2.1.1. Introduction

Throughout its history, intensive research within the RE discipline has motivated the development of a great number of approaches dedicated to dealing with requirements, as is the case of proposals which are goal-oriented [56], aspect-driven [46] or are based on requirements templates [47], to name but a few. On most occasions, these approaches use textual descriptions for the requirements specification, which are often gathered in non-formal models [16] or organized in requirements documents which are not usually formally structured [20]. In recent years, the growing impact of the Model Driven Engineering (MDE) approach [49], with proposals such as the OMG's Model Driven Architecture (MDA) [32] or Microsoft's Software Factories [23], has produced a change in the perspective in which these proposals consider requirements. In them, the entire development process is driven by models and, therefore, the requirements must also be represented as models through the use of a requirements metamodel which formally defines the concepts and relationships involved in the RE process [20].

Meanwhile, and bearing in mind that WE is a specific domain in which MDE can be successfully applied [41], the WE methodologies have been influenced by these approaches. This has motivated the most relevant methodologies (such as WebML, UWE or OO-H) to be aligned with MDE, and the development of WIS has been carried out by using different models and transformations among them.

This alignment of both the WE methodologies and the requirements management techniques with the model driven approaches suggests that the proposals developed with the aim of filling the gap related to requirements management in WIS should also be aligned with the model driven development paradigm [16]. With this trend in mind, the following subsections present our approach for a reinforcement of the requirements management activities in WIS.

2.1.2. Definition, concepts and related work

A requirements metamodel defines the concepts and relationships involved in the RE process in a formal manner. The advantages that a metamodel offers are numerous [32]. First, the metamodel defines both the elements that participate in the

requirements management process and the relationships between them in an unambiguous way. Moreover, it offers a formal basis upon which tools for (1) the management of the metamodel elements and (2) the definition of transformation rules from requirements to other elements can be constructed [34].

Several approaches for requirements metamodelling have recently appeared, but a reference model to deal with requirements does not exist [20]. To our knowledge, within the concrete scope of WE, the topic of requirements metamodelling has only been tackled in [16,7]. The requirements metamodel presented in [16] follows the design-oriented approach used by most WE approaches, meaning that the concepts that appear in it are very close to the WIS design (for instance, it includes visualization features or search activities as main constructs in the requirements elicitation activities). The main problem of this lack of high-level expressivity is that it is a generally avowed fact that design requirements are difficult to understand by stakeholders who are not directly related to the design. Such stakeholders require a more abstract way in which to express their own requirements, that is, a way that is closer to the domain under which the application is being developed.

This necessity of capturing higher-level communication goals and user requirements in a stable manner is noted in [7], in which a goal-oriented approach to model web requirements is used. The concept of goal was defined in the i* framework [57] and it models a high-level objective of one or more stakeholders. The concepts in the i* framework are very useful in the modelling of user goals, although their generality suggests the need to tailor them to specific domains. Following this trend, [7] uses i* as a basis, but specializes it in order to design a new requirements metamodel that collects particular WE concepts such as, for instance, a web requirements taxonomy.

As with the proposal of Escalona and Aragón [16], the requirements taxonomy proposed by Bolchini and Paolini [7] is similarly closely related to the way in which WE devised the application design at the time of its proposal, and the authors therefore note the need to improve automatic support and to deal with the concept of requirements reuse.

Other requirements metamodelling proposals which are not specifically focused on web-based systems also exist, such as COMET [4], a requirements modelling method that includes a requirements metamodel. However, this metamodel does not pay attention to non-functional requirements and it does not cover any method for requirements validation. REMM [54] presents another requirements metamodel but it does not consider the possibility of capturing the aforementioned high-level communication goals, nor does it cover methods for requirements validation. The concept of requirements reuse presented in [54] is, nevertheless, very useful and it can be adapted to the WIS scope.

In other cases, and since it is difficult to represent all the features included in the different approaches in a single metamodel, new requirements metamodels have been obtained by unifying the common concepts from the other proposals. Goknil et al. [20] follows this trend. However, the proposed metamodel does not consider important concepts such as that of requirements reuse and neither does it contain an automatic support to deal with the metamodel concepts.

To sum up, the existing requirements metamodels present interesting concepts, but there is a need for a metamodel which is independent of particular WE views, quality models or implementation technologies. We have therefore developed a requirements metamodel that synthesizes and simplifies the, from our point of view, most relevant concepts included in well-known RE proposals. Such simplification was necessary to avoid the burden of work usually added by more exhaustive RE practices, on the premise that, in the WE community, baselines must be generated very quickly [35] and therefore straightforward methods through which to gather requirements and connect them with validation methods are needed. This metamodel stresses the importance of performing requirements management activities as a first-order workflow for web methodologies and paves the way towards considering usability requirements from the first stages of the WIS development process. This metamodel is presented in Fig. 1 and is explained in more detail in the following subsection.

2.2. A requirements metamodel for WIS

Fig. 1 shows our approach for requirements metamodelling, which had as its origin an initial contribution to requirements metamodelling for WIS projects [40]. Let us now look at the concepts presented in Fig. 1 in greater depth. The key element in this metamodel is that of `requirement` which can be described by using a set of attributes (hidden in the figure) such as an `identifier`, its `type` and `priority` and its textual `description`. In order to avoid, as far as possible, the ambiguity inherent in natural language, the description of requirements can use `terms` included in a `glossary`. In Fig. 1, requirements can be classified as either `functional requirements` (what the system must do) and `non functional requirements` (how the system must do it). This classification is useful because, while functional requirements usually rely on `test cases` for their validation, non-functional

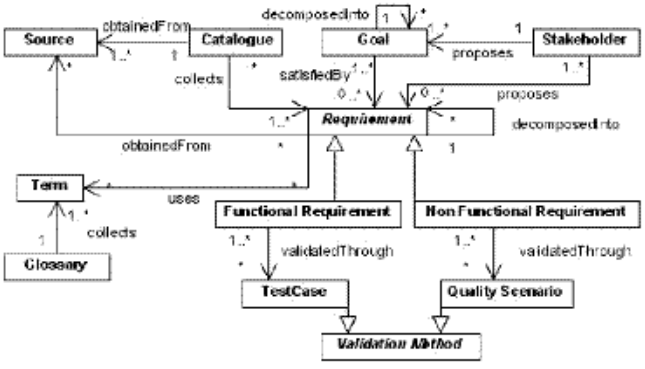

Fig. 1. A requirements metamodel for WIS.

requirements are related to `quality scenarios`. More complex requirements classifications have been avoided, as the possibilities are countless, and greatly depend on the designer's preferences. Requirements which are organized into tree-like structures (see e.g. quality models such as the ISO 9126 [27]), along with relationships between functional and non-functional requirements, can be defined in a simple manner by means of the unary relationship `decomposedInto` which is defined over the requirement concept.

Each requirement has a set of relationships with other elements. First, the metamodel includes the `goal` concept, borrowed from the goal-oriented RE, as a means to model the high-level objectives of one or more stakeholders. Each goal, which can be decomposed into subgoals, is related to the `stakeholder` (one or more) that proposes it, and is satisfied through the fulfilment of a set of requirements.

Fig. 1 also contemplates requirements reuse by introducing the `catalogue` concept, which represents a set of related requirements. In a nutshell, a catalogue puts together a set of requirements extracted from one or more sources (i.e., a law, an organization policy, a particular domain, a guideline, etc.) and can be reused in all the projects in which these sources are applicable. This concept has been successfully applied in traditional RE methods [52,53] and its adaptation may be useful in the context of web-based projects, where numerous concepts, such as standard quality models [27], web accessibility guidelines and laws [55,22], experts usability recommendations [43], etc. must receive attention. These requirements sources have been formalized in our proposal by using the `source` element. A catalogue of requirements can be extracted from each source. The concepts of catalogue and source have certain advantages. On the one hand, they help the stakeholders involved in a web-based development project to discover the numerous guidelines, laws, recommendations, etc. involved in the development. These guidelines are not always used and, on many occasions, inexperienced practitioners do not know all of them. On the other hand, when a web-based development project is forced to comply with a law or a guideline, practitioners need only go to the adequate catalogue and find the requirements with which their project must comply.

With the use of this metamodel, the stakeholders involved in the elicitation of requirements will have a better knowledge of the concepts that must receive attention during this stage. Moreover, the use of the tool that supports the metamodel (explained in Sections 4 and 5) will assist them to gather requirements and concepts, such as reuse, will allow them to obtain benefits with regard to their experience in previous projects. In Section 5 (see Fig. 6), an example of use which illustrates how the metamodel can be instantiated and can help stakeholders to put emphasis on the elicitation of requirements for their systems is shown.

3. Expressing usability requirements on WIS models for early usability evaluation

Usability is a critical quality attribute for the success of interactive software systems such as WIS [30]. A great number of studies have highlighted the benefits of considering usability such as, for example, the improvement of users' productivity or the reduction in training and documentation costs. Furthermore, the investments made in usability have been shown to return economic benefits [11], which has motivated important organizations such as IBM or Boeing Co. to consider usability as a relevant factor in their software products.

However, the consideration of usability in the development of software faces certain obstacles. As it was previously mentioned in Section 2, one of these is related to the absence of the consideration of usability requirements in early phases of the development lifecycle. In addition, Seffah and Metzker [50] highlights other problems, which are that the activities related to usability are usually decoupled from the mainstream software development process, the use of notations and tools with which to consider usability are different to those used in the development process, or the lack of automatic support for some usability activities.

An analysis of the methodologies and tools traditionally used for WIS development reveals that these obstacles continue to be present in the scope of WE. For example, these methodologies are not concerned with the consideration of quality attributes such as usability during the development process. This task is usually delayed until the system has been completely developed, using tools called *usability and accessibility validators* (see [28] for detailed information about these methods) which validate the HTML and CSS code of the WIS. Thus, the possibility of moving some of these validations towards earlier phases in the development cycle, such as design, and their integration in the methodologies, are not considered. Another example of the problems mentioned by Seffah and Metzker [50] can be observed in the use of quality metrics within the scope of WIS. Ruiz et al. [48] carried out a survey of these metrics and discovered that up to 385 quality metrics had been defined with the aim of measuring quality attributes over this kind of systems. Half of the defined metrics are related to usability but, as Ruiz et al. [48] argues, they are not usually defined in a precise way and are not supported by any tool or, if this support exists, it is offered by external tools which are independent of those used in the WIS development process.

While in Section 2 we emphasized the gathering of requirements, such as usability requirements, in early phases of the development cycle as a means to obtain numerous benefits, the following sections show an approach with which to consider the definition and evaluation of usability requirements and metrics over the conceptual models used by WIS development methodologies. This approach arose after discovering that, during the elicitation of WIS requirements, stakeholders had proposed certain usability requirements than could not be expressed over the models used in WIS development. Thus, an extension of the navigational models that permits the expression of those requirements which cannot, at present, be expressed in design time has been defined. A study of the usability metrics that can be evaluated over WIS with the aim of improving its quality has also been carried out, along with the development of a prototype tool which permits the evaluation of the aforementioned usability requirements and metrics, and which can be integrated in the CAWE tools used by WIS development methodologies.

3.1. Usability on navigational models

Software usability is a quality attribute for which it is difficult to find a standard definition. The ISO 9126 standard [27] defines it as "the capability of the software product to be understood, learned, used and attractive to the user, when used under specified conditions". Nielsen [42] defines it as "the learnability and memorability of a software system, its efficiency of use, its ability to avoid and manage user errors and user satisfaction" and the ISO 9241-11 standard [26] as "the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use".

These multiple definitions imply that usability has been interpreted from different perspectives and that different stakeholders perceive it in different ways [50]. These definitions also show that usability is an abstract attribute which depends on numerous factors, meaning that if we wish to improve the usability of a WIS, we must pay attention to multiple features. The approach presented in this paper centres on usability as it is perceived by an end user, that is, as a quality attribute which allows end users to perform

the expected tasks more efficiently. Although this definition is a little more precise than those previously mentioned, it continues to affect numerous features such as the intuitive navigation in the system, ease of use, simplicity in carrying out tasks or a comfortable and attractive presentation. Our approach is centred on improving the usability of a WIS by improving the navigation and the access to the information and functionalities that it presents, that is, by offering an intuitive navigation that makes it easy for the users for whom the WIS was conceived to carry out their tasks.

The approach was developed by studying the existing methodologies for WIS development and the models used by them to develop systems. Although each methodology presents specific features, in general, all of them use different models, such as navigational models (to model the interaction between users and the WIS), behaviour models (to model the data and functionality offered by the system) and presentation models (to represent features related to the final presentation of the system). By following a model driven approach, these models drive the development process and serve as a basis for the construction of the final WIS.

Our approach is centred on navigational models. Although the navigational models used by each methodology present slight differences [18], they are usually composed of two main elements: nodes and links between nodes. A node is used to represent a set of information or functionality that will be presented to WIS users. Links are used to join nodes, indicating the possibility of navigating from one node which represents a piece of information or a functionality to another. Other navigation structures such as menus or indexes may also appear in these models. Fig. 2 shows an example which may be useful in the observation of these concepts. It is a fragment of a navigational model for a WIS through which to purchase books via the Internet, and it presents information and functionalities related to their purchase.

The quality of navigational models is important since these models represent the possible paths that users may follow whilst navigating via WIS. Thus, errors in these models or less useful navigation designs influence the usability of the WIS which is finally developed. A number of studies centred on the evaluation and improvement of the quality of these models have been carried out. For example, Abrahão et al. [1] defines a set of metrics to provide modellers with information about the quality of navigational models, and Molina et al. [39] proposes a similar strategy for the verification and demonstration of properties of navigational models. Dhyani et al. [10] and Ruiz et al. [48] analyse and summarize the research carried out by numerous authors with regard to the definition of metrics to evaluate the quality of the different artefacts that participate in WIS development, but note that one of the main problems of these metrics is that they do not have automatic support or that it is offered by isolated tools which are not integrated in WIS development tools.

Our approach uses the positive features of the aforementioned proposals and additionally complements them with two improvements. Firstly, it not only allows modellers to use predefined metrics, but also permits them to use their own defined usability requirements and metrics over navigational models. Secondly, it offers an automatic support for these requirements which, when integrated in WIS development tools, may permit modellers a comfortable means through which to take advantage of the approach. The following subsection gives further details of how the first improvement can be carried out.

3.1.1. Evaluating usability requirements in modelling time

During the analysis of the requirements elicitation process for WIS carried out to develop the proposal presented in Section 2, we discovered that the stakeholders involved in the WIS development had proposed certain usability requirements which could not be expressed over the models used by WIS methodologies to build the system. The existence of this kind of requirements has been stressed in [3] and the application of the approach presented in Section 2 allowed us to discover some of them.

Some of these requirements were related to access to WIS information and functionalities. For example, some stakeholders wished to establish the maximum number of clicks that an end

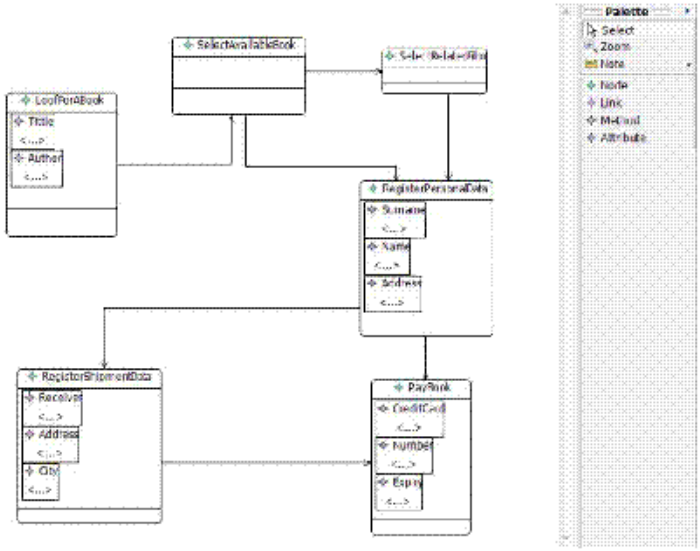

Fig. 2. An example of a navigational model.

user would need to make in order to carry out a concrete task. On other occasions, they wanted to express that not all the information and functionalities had the same importance, bearing in mind that primary information and functionalities related to the main aim of the system, and other information or functionalities that can be considered as secondary, usually exist in a WIS. For example, in the WIS through which to purchase books shown in Fig. 2, the primary functionality must be that related to searching for books and buying them. Other information or functionalities such as, for example, buying a DVD associated with the book may also be shown. This information is important but it is not the main aim of the system. In these cases, the stakeholders wanted this primary information to be close to the end users, for example, in terms of proximity to the entry point of the WIS.

On other occasions, the modellers wished to establish constraints relating to the order in which users visit nodes, or they wished to force the existence of direct or indirect connectivity among the nodes that represent related functionalities, under the premise that this facilitates navigation through the system, thus improving its usability. Table 1 summarizes some of these requirements.

Since this kind of requirements cannot be expressed during WIS modelling, they are usually forgotten and are not taken into account during the development process. This may mean that the WIS which is finally developed does not permit an intuitive navigation, and may lead users to feel lost and disoriented. In addition, the opportunity of obtaining the aforementioned benefits of usability analysis based on models is missed. To attempt to fill this gap, we have developed an extension of the navigational models which permits the expression and evaluation of this kind of requirements in modelling time. We will use the set of requirements shown in Table 1 to illustrate our approach, which is explained in the following subsection.

3.1.2. Extending navigational metamodels with usability features

The solution to the problem of allowing modellers to express the aforementioned usability requirements over their models consists of extending the metamodel of the navigational models with those entities, relationships and attributes that will allow this information to be collected. As was mentioned in Section 3.1, the two main elements of which navigational models are composed are nodes and links. We shall use these elements to illustrate our approach, considering a simplified fragment of the navigational metamodel (see top left of Fig. 3).

Support has been offered to the kind of requirements shown in Table 1, by extending the metamodel of navigational models in the following manner:

- The addition of new attributes. These attributes (called *MaxDistance* and *MinDistance*) are added to the element *NavigationNode* and are used to support the requirements related to distances between nodes.
- The addition of new links. Two recursive links have been added for navigation nodes (called *previousNodes* and *laterNodes*). These links represent, for each node, a list of previous and later nodes that a user must also visit if s/he visits that node and which are useful in supporting the kind of requirements related to navigation constraints.
- The addition of new elements. The *Level* entity and its attributes are used to support the requirements related to the importance of functionality and information. This entity represents the concept of *importance of a node* and its link with the *NavigationNode* entity allows modellers to label each node with an importance level. Each *Level* has three attributes: a name and the integers *MaxLevelDistance* and *MinLevelDistance*, which define the minimum and maximum distance between the nodes labelled with a Level and the node that represents the entry point to the WIS.

The extended metamodel obtained after adding these elements is shown at the bottom of Fig. 3. It is important to emphasize that the set of requirements and the extension of navigational metamodels presented are used to illustrate our proposal, but obviously other requirements which require new extensions can also be discovered and the approach could even be utilized on other models used in WIS development (e.g. presentation models) and not only on those which are navigational. In this case, we shall follow the same strategy, which consists of adding the attributes, entities and relationships that allow the expression of these new requirements over the models involved.

After analysing this extension and exploring its technical details in greater depth, we observed that various solutions could be used to obtain the extended metamodel:

(i) A direct extension of the navigational models proposed by WIS methodologies. In this case, the metamodel of the navigational models will be taken as the entry (see top left of Fig. 3), and all the new elements will be added to it. The result of this option is shown at the bottom of Fig. 3.
(ii) The creation of an independent metamodel that unites those elements (attributes, entities and relationships) which are necessary to support the new requirements (see top right of Fig. 3), which is then combined with the original navigational metamodel (see top left of Fig. 3) by using model transformations.

Table 1
A set of usability requirements that cannot be expressed in navigational models.

Goal	Requirements	Rationale
Definition of importance levels	Information/functionalities in the WIS must be organized into various levels	Not all the information/functionalities in the WIS have the same importance
Distances between nodes	To define Max/Min distance from the entry point to the WIS for each importance level	Possibility of detecting nodes labelled as important too far from the entry point and nodes labelled as less important excessively near
	To define Max/Min distance from the entry point to the WIS for each node	Possibility of detecting nodes which are little accessible for users
	To define distances between given nodes	If two nodes represent information or functionalities related to each other, modellers probably wish that they will be near
Connectivity constraints	To establish direct connectivity between nodes	It may be useful to express the requirement that two nodes must be directly connected
	To establish indirect connectivity between nodes	It may be useful to express that a node must be reachable from another one
Navigation constraints	To force previous crossing by a node	For each node it is possible to define a set of nodes that must be previously visited before reaching it
	To force later crossing by a node	For each node it is possible to define a set of nodes that must be visited after visiting it

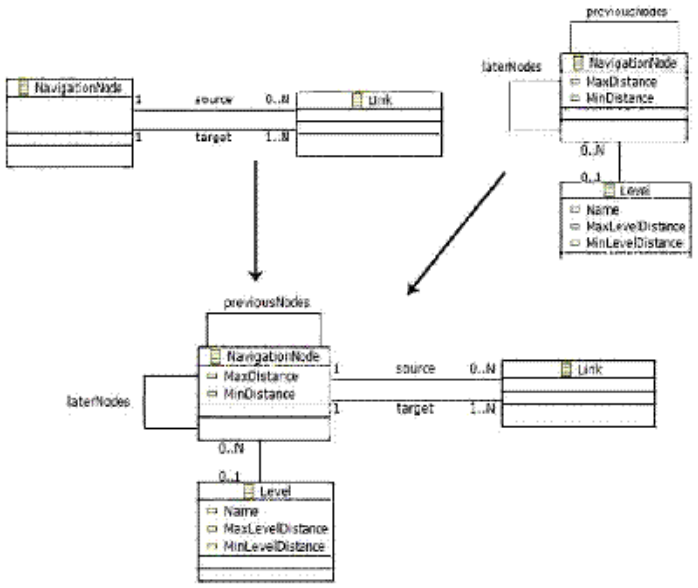

Fig. 3. Two options with which to extend navigational metamodels.

For our purposes, we have chosen the second option. The main problem of the first option is that it creates a hard coupling between the elements of which the navigational metamodels were originally composed and the new added elements. This might limit the extensibility of the proposal and it would be difficult for modellers to distinguish between the elements of the original metamodel and the elements added to support usability features. On the contrary, in the second option the original metamodels and the extensions used to consider usability features over them remain decoupled. It facilitates the possible extension of the approach because, to support new requirements, we only need to focus on the metamodel that combines the new elements. As exchange, it will be necessary to define the transformation which will obtain the target metamodel from the separate metamodels. This means of solving the merging of models and metamodels corresponds to a special kind of model transformations called model weaving [5]. Model weaving tackles the problem of given a model m_a which conforms to a metamodel M_a and a model m_b which confors to a metamodel M_b, then an integrated model m_{ab} can be obtained which conforms to a woven metamodel M_{ab}. As the reader may observe, this situation is similar to that presented above: we have two metamodels (the original navigational metamodel (M_a) and the metamodel created in order to consider usability features (M_b), which are shown at the top of Fig. 3) and we are interested in obtaining navigational models with usability features, which must conform the metamodel shown at the bottom of Fig. 3.

The weaving transformations can be specified in any transformation language such as, for example, QVT [44], the OMG standard for this aim within the scope of the MDE paradigm. Fig. 4 shows a fragment of the transformations that permit weaving between models m_a and m_b (that is, a navigational model and a usability features model which conform to their respective metamodels presented at the top part of Fig. 3) in order to obtain the model m_{ab}. These transformations have been defined by using the QVT *Relations* language, one of the languages defined in the QVT specification. Owing to lack of space, we can only show how the main entities of each metamodel are joined together in the woven metamodel (M_{ab}). These are:

- From M_b NavigationNode to M_{ab} NavigationNode (Fig. 4a). This QVT relation transfers the navigation nodes of a usability model ("nn" in Fig. 4a) to the target model, from now on called the weaving model ("wnn" in figure), using the attribute values of "nn".
- From M_b Level to M_{ab} Level (Fig. 4b). As in the previous relation the Level entities in the usability model are transferred into the weaving model.
- From M_a Link to M_{ab} Link (Fig. 4c). Finally, the Link entity of the navigational model is created in the weaving model. This relation selects a link in the navigational model and the references to its source and target node, and then matches the nodes in the weaving model that will be used to create the new link in the weaving model. This pattern matching is defined by using the variables "snnn" and "tnnn", which are bound to the adequate values in the models involved in the transformation. Moreover, we use the *when* clause in the relation to ensure that the two navigation nodes involved in the link have already been created in the weaving model.

Finally, it is important to highlight that the transformation presented here is not complete, since other features of the original model (such as associations, etc.) should also be transferred to the weaving model. However these other relations are not shown due to lack of space.

After the model weaving, it is time to evaluate the fulfilment of the usability requirements that have been expressed over the models built. With this aim, and using the OCL (Object Constraint Lan-

```
(a)
top relation QualityNavigationNodesToWeaving {
 nnn: String; maxD, minD: Integer;
 checkonly domain usability nn : NavigationNode {
   Name = nnn,
   MaxDistance = maxD, MinDistance = minD };
 enforce domain weavingMM wnn : NavigationNode {
   Name = nnn,
   MaxDistance = maxD, MinDistance = minD };
}

(b)
top relation LevelToWeaving {
  ln: String; maxD, minD: Integer;
  checkonly domain usability l:Level {
     Name = ln, MaxLevelDistance = maxD,
     MinLevelDistance = minD  };
  enforce domain weavingMM lw:Level{
     Name = ln, MaxLevelDistance = maxD,
     MinLevelDistance = minD  };
}

(c)
relation LinkToWeaving {
 snnn, tnnn: String;
 checkonly domain navigational l : Link {
  source = snn:NavigationNode { name = snnn }
  target = tnn:NavigationNode { name = tnnn }
 };
 checkonly domain weavingMM
  swnn : NavigationNode { Name = snnn };
 checkonly domain weavingMM
  twnn : NavigationNode { Name = tnnn };
 enforce domain weavingMM l2 : Link {
  source = swnn, target = twnn };
when {
  QualityNavigationNodetoWeaving(snn,swnn);
  QualityNavigationNodetoWeaving(tnn,twnn);
 }
}
```

Fig. 4. A fragment of the model weaving transformations.

guage) plug-in [14] of the Eclipse platform, a set of OCL constraints has been implemented to guarantee that the usability features expressed are fulfilled. If any constraint is violated, modellers will be notified and will be able to modify their models in order to solve the detected problems.

3.1.3. Usability metrics for navigational models

Since the quality of navigational models influences the quality of the systems finally developed, the WIS research community has defined a number of quality metrics with the aim of improving the quality of these models. As our approach permits the definition and execution of OCL expressions over navigational models, an additional benefit of its application is that it can be used as an integrated framework for the definition and evaluation of usability metrics, which are usually defined in multiple isolated tools that are not integrated in the WIS development process. The WQM (Web Quality Metrics) model [48] was used to discover a set of evaluable metrics, and this carried out a review of literature in which it found up to 385 metrics defined to evaluate the quality of the artefacts used in WIS development. An analysis of these metrics has been carried out with the aim of selecting those that fulfil two requirements: (i) they must be related to the evaluation of WIS usability and (ii) it must be possible to evaluate them in modelling time, since our approach attempts to take advantage of the benefits of the model based usability analysis.

Among the metrics defined in WQM, 48% are related to usability and we have selected those which, since they are measurable in modelling time, are related to navigation through the system. Some of the usability metrics such as, for example, those that measure the number of broken links in the WIS finally developed, that count the number of graphics or are related to the navigation effort cannot be evaluated over models, since sufficient information for this purpose does not exist in modelling time. Nevertheless, a set of 50 quality metrics that can be defined and evaluated over navigational models already exist. The rationale behind these metrics is not within the scope of this work but more information is available in [48]. Lack of space prevents us from offering an exhaustive list of these metrics, but some examples are shown in Table 2. As we can see, these metrics offer quantitative information about the models (number of nodes and links, depth and width of the model, etc.), information concerning the connectivity among the elements in the model (relationship between the number of nodes and links, number of in and out links), etc.

After identifying these metrics, our next aim was to define and integrate them within the tool that supports the whole approach, as had been done with the usability requirements. This is shown in Section 5.2 as well as how the tool offers modellers the possibility of defining their own metrics and queries over their models by using OCL expressions.

3.2. Approach overview

Fig. 5 shows an overview of our approach using the SPEM notation [45]. It starts with the systematic elicitation of requirements using the proposed requirements metamodel and emphasizes the gathering of quality requirements such as those related to usability. After this phase, the requirements obtained must be expressed over the following models used to build the system. In our case, we focus on navigational models, which are built together with the usability model that offers modellers the capacity to express usability requirements which cannot at present be defined over these models. In the weaving phase, both models are combined to obtain navigational models extended with usability features. After the system has been designed, the fulfilment of the usability requirements is checked by means of the execution of the OCL constraints which are predefined with this aim. Moreover, in this phase modellers execute the implemented usability metrics over their models in order to obtain information about their quality. The information obtained after checking the requirements and evaluating the metrics is used by modellers to, if necessary, improve the models in their system. Finally, the models obtained will be used to implement the system.

Table 2
Some examples of usability metrics for navigational models.

Goal	Examples of metrics
Quantitative information	Depth, width, diameter, radius, number of nodes, number of links, number of attributes and methods, etc.
Model connectivity	Number of in and out links, compactness, structure, average connected distance, connectivity density, etc.

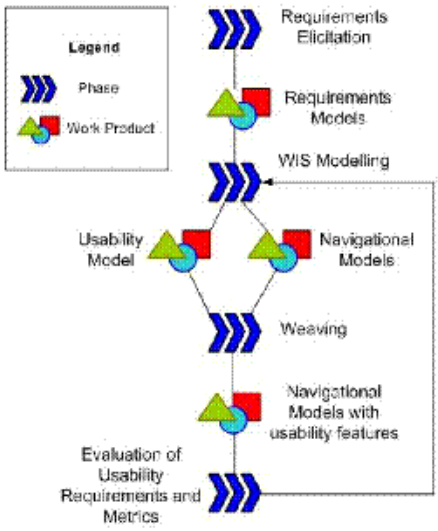

Fig. 5. Approach overview.

4. Automatic support

As was previously mentioned, one of the key factors for the successful handling of usability in the WIS development process is the existence of an automatic support for the approach which can be simultaneously integrated in the tools usually used in the WIS development. We have, therefore, designed a tool which supports the presented contributions and allows stakeholders to manage the concepts involved in a comfortable manner. The following sections explain the considerations taken into account when choosing the appropriate technological space in which to develop this tool.

4.1. Technological environment

Certain considerations had to be taken into account when selecting the technological environment in which this tool would be implanted. First, it was necessary to use an environment that would permitted the easy definition of metamodels, since our approach involves both a requirements metamodel and the extensions carried out to consider usability on navigational models. Furthermore, as we wished various stakeholders (and not only designers) to use our approach, we needed to offer them a graphical tool with a usable and comfortable interface which would allow them to create and manipulate models compliant with our approach and which would also facilitate the evaluation of the quality of their models. Furthermore, the presented approach was not thought to be an isolated effort but an effort oriented towards its integration in WIS development processes with the aim of reinforcing both the requirements management and usability evaluation activities. Therefore, the capacities offered by the technological environment in order to extend the tool with new functionalities or to integrate it in existing tools used in WIS development had to be considered.

Given these premises, the Eclipse platform and, in particular, the Eclipse Modelling Framework (EMF [12]) was selected. Eclipse and EMF offer certain suitable features that make them interesting for our approach. First, EMF offers support to deal with MOF, the standard that OMG recommends for describing metamodels. EMF is therefore useful for the creation and manipulation of models and metamodels such as those proposed in our approach. Second, it is an open source platform-independent project and the architecture, which is based on plug-ins, makes it easy to reuse and to add functionality. Moreover, Eclipse offers an OCL implementation which is useful in defining requirements and metrics in this language. Finally, a further important advantage of Eclipse is that some CAWE tools used for WIS development such as WebRatio [2] are being migrated to this platform and some of the efforts [38] to reach an agreement with regard to the concepts managed in WE use Eclipse as technological environment. The integration of our automatic support in these Eclipse tools through a new plug-in can be directly obtained.

The steps carried out for the implementation of this automatic support have been the following:

- Definition of the metamodels that allow both the requirements modelling and the design of navigational models with usability features (see Figs. 1 and 3), using the EMF project.
- Definition of graphical editors that allow stakeholders to define requirements models and navigational models with usability features in an easy and intuitive manner, by using a palette of elements similar to those included in any CASE tool. The models that can be defined with these editors conform to the metamodels defined in the previous step. The GMF (Graphical Modelling Framework [13]) project has been used for this purpose.
- Definition of the weaving transformation explained in Section 3.1.2 using Medini QVT [51].
- Definition and implementation of the checks that support the evaluation of usability requirements and metrics. This task is supported through the implementation of OCL offered by Eclipse [14].

The following section will show the appearance and functionality of this tool through an example of its use.

5. Case study

This section illustrates how the presented approach can be used for the elicitation of requirements and the subsequent evaluation of usability attributes in design time through an example of its use which corresponds to a simplified on-line book sale system. The example has been developed using the automatic support implemented to support our approach.

For the sake of simplicity, let us assume that the CEO of a certain company has decided to expand her/his business strategy, and that her/his goal is to *attract new clients*. The CEO believes that offering the books through Internet may positively influence both the attainment of new clients and the reduction of the cost associated with the sales process, so s/he has embarked on a web book sales system development process. Both the CEO and the web customer who will interact with the application have as their main functional requirement *buy books*. This requirement can be decomposed into two functional requirements: *browse available books* and *purchase books*.

In addition, several non-functional requirements that the web based system must fulfil have been identified. Firstly, the buy books functionality should follow *accessibility* guidelines to allow web customers with disabilities to access the system according to the related laws and guidelines created for this aim such as Section 508 [22] or WAI recommendations [55]. Additionally, and in relation to usability attributes, the system should provide *information accuracy* while browsing through the available books: synopsis, prices and so forth should be reliable. Also, the application

learnability should be high, that is, the application should be simple enough for novice users to easily learn its operation. With regard to navigation through the WIS, some requirements have been established. The CEO wants to offer, if they exist, the possibility of buying the films associated with each book, but s/he considers this information as secondary, so it must not be directly accessible from the entry point. The browse books functionality is the most important, so it must be at a maximum distance of one click from the entry point. In addition, it must not be possible to buy books if the user has not previously registered in the system and the functionalities to select a book and pay for it must be directly attainable. Finally, the purchase process should be performed assuring the security of the customer data.

5.1. Instantiating the requirements metamodel

If we check the metaclasses of the metamodel in Fig. 1, we can observe how:

(i) The CEO and web customer both instantiate the *Stakeholder* metaclass.
(ii) Attract new clients instantiates the *Goal* metaclass.
(iii) Buy books, browse available books or purchase books are all *Functional Requirement* occurrences.
(iv) The requirements related to accessibility, ease of navigation, learnability, security and information accuracy are all instances of *Non-Functional Requirement* instances.
(v) The WAI guidelines are a requirements *source*. If this source has been applied to other projects, the associated requirements can be directly incorporated into this system by using the corresponding requirements *catalogue*.

Logically, these requirements are decomposed into other more concrete which have not been shown for the sake of simplicity. The requirements metamodel significantly helps in the systematic identification of these requirements which in its turn serves as a good baseline to start the WIS modelling. These identified requirements must then be reflected on WIS models, such as content, navigation or presentation models.

Fig. 6 shows the automatic support for the requirements metamodel. After formalizing it by using EMF, a graphical editor which permits the construction of requirements models that conforms to the metamodel has been developed. On the right side, the tool shows a palette with which to manage the concepts involved (stakeholders, goals, requirements, etc.), and this is next to an intuitive icon which is used to build the requirements models. In the figure, only a fragment of the application example is shown, but the complete model would of course include more requirements, along with other stakeholders, sources, etc. Until now, the goal proposed for the CEO of the company has been expressed together with some of the requirements necessary to fulfil them. The `Navigation constraints` requirement must be divided into the other requirements related to navigation mentioned in the description of the application example. Since the WAI recommendations are used in multiple projects, it is possible to take advantage of the concept of reuse thanks to the requirement catalogue presented in the model.

5.2. Designing navigational models with usability features

After the requirements elicitation phase, the following step consists of building the conceptual models that express the gathered requirements. Further details of how the requirements related to ease of navigation can be represented on our navigational models extension and how the quality of these models can be evaluated using the usability metrics defined over them will now be given.

As occurred with the requirements metamodel, EMF and GMF have been used to define the metamodels for the navigational models and also to build the graphical editors that allow modellers to define these models comfortably. Fig. 2 shows a fragment of the navigation model for our example application, which has been modelled using the graphical editor.

Since these models have been extended to support usability features, modellers can express the requirements related to the navigation while they simultaneously build their navigational models. Fig. 7 shows how some requirements constraints can be defined in the usability features model. As the CEO established, there is information concerning two levels of importance, defined as `High` and `Low`, in Fig. 7. The nodes related to the sales of books have been labelled with the `High` Level and those related to the sales of films with the `Low` Level. Moreover, certain maximum/minimum distances have been defined for these nodes. As we can observe at

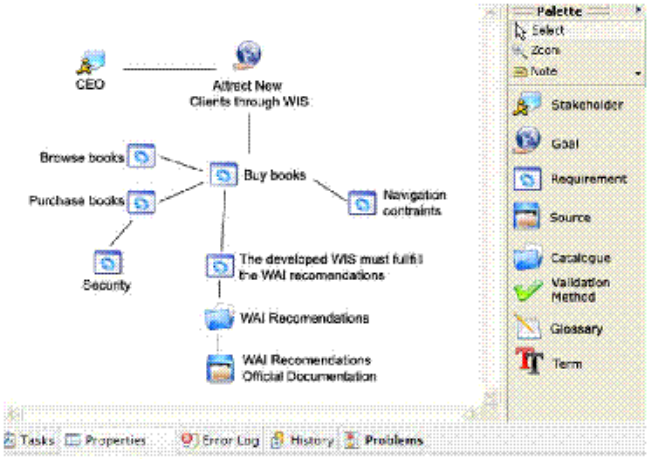

Fig. 6. A fragment of the requirements model for the case study.

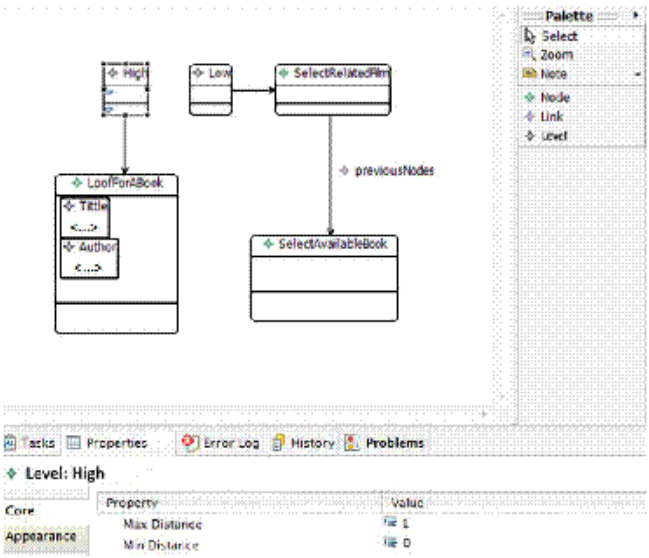

Fig. 7. Adding usability constraints to the navigational model.

the bottom of Fig. 7, the modeller has defined a maximum distance of one click among the nodes labelled with `High` and the entry point.

After their expression of these usability requirements, modellers can execute the proofs implemented to assure that these requirements are fulfilled. Furthermore, modellers also have at their disposal the implementation of the set of usability metrics mentioned in Section 3.1.3. The information obtained after execution provides them with valuable information with which to evaluate the quality of their models, and to improve them if necessary. Furthermore, the OCL plug-in allows modellers to define their own metrics over their models. Fig. 8 shows some straightforward examples of these metrics.

Fig. 8a shows an example of the kind of usability metrics shown in Table 1. In this case, it counts the number of nodes in the model. Fig. 8b is useful to illustrate one of the metrics that the modeller can define and execute over their models. In this case, the OCL constraint is defined over the model of Fig. 7 and selects those nodes labelled with a `High` importance level. Finally, it is important to emphasize that modellers can define not only metrics but even other kind of constraints over their models. For example, Fig. 8c illustrates a constraint that establishes that all the links in the model must have different names. These metrics and constraints can be incorporated in the tool and the modellers just select them and obtain information that can use to improve their models.

```
(a) self.nodes->size()
(b) self.nodes->select(n:Node | n.level.Name = "High")
(c) self.links->isUnique(l: link | l.name)
```

Fig. 8. Some examples of OCL metrics and constraints.

6. Conclusions and further work

The improvement of the quality of WIS must start from the early stages of the development lifecycle. Our approach makes it possible to deal with usability attributes from the requirements elicitation and early WIS design phases. It provides a requirements metamodel which helps in the elicitation of WIS requirements and complements the other models used by WE methodologies, which, until now, often began their development process from WIS design. After using this metamodel, more attention was paid to requirements elicitation, and a set of usability requirements which can be evaluated in design time has been discovered. These requirements have served as a basis to develop an extensible approach that allows modellers to define and evaluate usability requirements in modelling time.

Moreover, after studying the relevant literature, a set of usability metrics which can be evaluated over models have been identified and integrated in our approach, which now offers an integrated framework for these metrics. These help modellers to improve the quality of their models and avoid the necessity of using multiple isolated tools which only implement subgroups of these metrics.

As further work, and with regard to the reinforcement of the requirements management activities in WIS methodologies, we are conscious that our metamodel is a first step towards establishing the basis for new improvements. We are therefore refining it, and simultaneously developing a profile of the i* notation as a means to reduce the complexity of object models when the size of the WIS projects grows. In addition, the connection of this metamodel with validation methods that help to validate the fulfilment of the collected requirements and the definition of transformation rules from requirements to other artefacts that participate in WIS development, such as navigational or presentation models, is being carried out. With regard to the activities of early usability evaluation, the approach can be extended in order to discover and incorporate more requirements and metrics which can be evaluated in design time, thus enriching the navigational models' quality extension. Moreover, as was mentioned in Section 3, a similar approach can be carried out on other models used in WIS development, such as presentation models. A set of experiments to help provide a more empirical evaluation of the proposal is also being defined.

References

[1] Abrahão S, Condori N, Olsina L, Pastor O. Defining and validating metrics for navigational models. In: METRICS 2003, Sydney, Australia. IEEE Press; 2003. p. 200–10.

[2] Acerbis R, Bongio A, Brambilla M, Butti S. WebRatio 5: an eclipse based case tool for engineering web applications. In: ICWE; 2007. p. 501–5.

[3] Atterer R, Schmidt A. Extending web engineering models and tools for an automatic usability validation. J Web Eng 2006;1:43–64.

[4] Berre AJ. Camex (component and model based development methodology); 2006. <http://modelbased.net/camex/>.

[5] Bézivin J. On the unification power of models. Softw Syst Model 2005;4(2):171–88.

[6] Blaine JD, Cleland Huang J. Software quality requirements: how to balance competing priorities. IEEE Softw 2008;25(2):22–4.

[7] Bolchini D, Paolini P. Goal driven requirements analysis for hypermedia intensive web applications. Requirements Eng J 2004;9(2):85–103.

[8] Ceri S, Fraternali P, Bongio A, Brambilla M, Comai S, Matera M. Designing data intensive web applications. San Francisco, CA, USA: Morgan Kaufmann Inc.; 2002.

[9] Damian D, Chisan J. An empirical study of the complex relationships between requirements engineering processes and other processes that lead to payoffs in productivity quality and risk management. IEEE Trans Softw Eng 2006;32(7):433–53.

[10] Dhyani D, Keang W, Bhawmick S. A survey of web metrics. ACM Comput Surv 2002;34(4):469–503.

[11] Donahue GM. Usability and the bottom line. IEEE Softw 2001;18(1).

[12] Eclipse. Eclipse modeling framework project (EMF); 2007. <http://www.eclipse.org/modeling/emf/>.

[13] Eclipse. Eclipse graphical modeling framework; 2008. <http://www.eclipse.org/gmf/>.

[14] Eclipse OCL Model development tools (MDT); 2008. <http://www.eclipse.org/modeling/mdt/?project=ocl>.

[15] Egner M. Poor project management number one problem of outsourced e projects. Research Briefs, Cutter Consortium; 2000.

[16] Escalona MJ, Aragón G. NDT a model driven approach for web requirements. IEEE Trans Softw Eng 2008;34(3):377–90.

[17] Escalona MJ, Koch N. Requirements engineering for web applications: a comparative study. J Web Eng 2004;3:193–212.

[18] Escalona MJ, Torres J, Risaca M, Gutiérrez JJ, Villadiega D. The treatment of navigation in web engineering. Adv Eng Softw 2007;38(4):267–82.

[19] Ginige A, Murugesan S. Guest editors' introduction: web engineering—an introduction. IEEE MultiMedia 2001;8(1):14–8.

[20] Gaknil A, Kurev I, van den Berg K. A metamodeling approach for reasoning about requirements. In: ECMDA-FA; 2008. p. 310–25.

[21] Gómez J, Cachero C, Pastor O. Conceptual modeling of device independent web applications: towards a web engineering approach. IEEE Multimedia 2001;8:26–39.

[22] USA Government. Section 508: the road to accessibility; 2007. <http://www.section508.gov/>.

[23] Greenfield J, Short K, Cook S, Kent S. Software factories: assembling applications with patterns models frameworks and tools. John Wiley & Sons; 2004.

[24] Standish Group. The chaos report; 2002. <http://www.standishgroup.com/>.

[25] Insfran E, Fernandez A. A systematic review of usability evaluation in web development. In: WISE workshops. LNCS, vol. 5176; 2008. p. 81–91.

[26] I.S.O. Iso 9241 11. Ergonomic requirements for office work with visual display terminals (vdts) – part 11: guidance on usability; 1998.

[27] I.S.O. Iso/iec 9126 1. Softw. eng. product quality – part 1: quality model; 2000.

[28] Ivory MY, Hearst M. An empirical foundation for automated web interface evaluation. PhD thesis, University of California at Berkeley; 2001.

[29] Juristo N, Moreno AM, Isabel Sánchez M. Analysing the impact of usability on software design. J Syst Softw 2007;80(9):1506–16.

[30] Juristo N, Moreno AM, Sanchez-Segura MI. Guidelines for eliciting usability functionalities. IEEE Trans Softw Eng 2007;33(11):744–58.

[31] Kappel G, Michlmayr E, Pröll B, Reich S, Retschitzegger W. Web engineering old wine in new bottles? In: ICWE; 2004. p. 6–12.

[32] Kleppe AG, Warmer J, Bast W. MDA explained: the model driven architecture: practice and promise. Boston, MA, USA: Addison Wesley Longman Publishing Co., Inc.; 2003.

[33] Koch N, Kraus A. The expressive power of uml based web engineering. In: Second international workshop on web oriented software technology (IWWOST); 2002.

[34] Koch N, Zhang G, Escalona MJ. Model transformations from requirements to web system design. In: ICWE; 2006. p. 281–8.

[35] Lang M, Fitzgerald B. Web based systems design: a study of contemporary practices and an exploratory framework based on method in action. Requirements Eng J 2007;12(4):203–20.

[36] Lowe D. Web system requirements: an overview. Requirements Eng J 2003;8(2):102–13.

[37] Matera M, Rizzo F, Carughi GT, Perrella A. The usability dimension in the development of web applications. In: Handbook of research on web information systems quality. Idea Group Publishing; 2008. p. 238–49.

[38] MDWEnet Initiative. Web engineering, metamodel; 2008. <http://www.pst.ifi.lmu.de/projekte/mdwenet/index>.

[39] Molina F, Lucas FJ, Toval A, Vara JM, Cáceres P, Marcos E. Towards quality web information systems through precise model driven development. In: Handbook of research on web information systems quality. Idea Group Publishing; 2007. p. 317–55.

[40] Molina F, Pardillo J, Toval A. Modelling web based systems requirements using WRM. In: II international workshop on web usability and accessibility. WISE workshops. LNCS, vol. 5176; 2008. p. 122–31.

[41] Moreno N, Romero JR, Vallecillo A. An overview of model driven web engineering and the MDA. In: Web engineering and web applications design methods; 2007 [chapter 12].

[42] Nielsen J. Usability engineering. Boston, MA, USA: AP Professional; 1993.

[43] Nielsen J. Designing web usability: the practice of simplicity. New Riders Publishing; 2000.

[44] OMG. MOF QVT standard specification; 2005. <http://www.omg.org/docs/ptc/05-11-01.pdf>.

[45] OMG. Object management group. Software process engineering meta model v1.1; 2005. <http://www.omg.org>.

[46] Rashid A, Moreira A, Tekinerdogan B. Special issue on early aspects: aspect oriented requirements engineering and architecture design. IEE Proc Softw 2004;151(4):193–6.

[47] Robertson S, Robertson J. Mastering the requirement process. Boston, MA, USA: Addison Wesley; 1999.

[48] Ruiz J, Calero C, Piattini M. Classifying web metrics. In: Software audit and metrics; 2004. p. 22–37.

[49] Schmidt DC. Guest editor's introduction: model driven engineering. IEEE Comput 2006;39(2):25–31.

[50] Seffah A, Metzker E. The obstacles and myths of usability and software engineering. Commun ACM 2004;47(12):71–6.

[51] Ikv++ Technologies. Medini QVT 1.4; 2008. <http://www.ikv.de/>.

[52] Toval A, Moros B, Nicolás J, Lasheras J. Eight key issues for an effective reuse based requirements process. J CSSE 2008;23(5).

[53] Toval A, Nicolás J, Moros B, García F. Requirements reuse for improving information system security: a practitioner's approach. Requirements Eng J 2002;6(4):205–19.

[54] Vicente Chicote C, Moros B, Toval A. Remm studio: an integrated model driven environment for requirements specification, validation and formatting. J Object Technol 2007;6(9):437–54 [special issue TOOLS EUROPE 2007].

[55] World Wide Web Committee (W3C). Web accessibility initiative (WAI); <http://www.w3.org/wai/>.

[56] Yamamoto S, Kaiya H, Cox K, Bleistein SJ. Goal oriented requirements engineering: trends and issues. IEICE Trans 2006;89-D(11):2701–11.

[57] Yu ESK. Towards modelling and reasoning support for early phase requirements engineering. In: RE, IEEE Computer Society; 1997. p. 226–35.

Artículo en *International Journal of Intelligent Systems*

An MDE Modelling Framework for Measurable Goal-oriented Requirements*

Fernando Molina[1], Jesús Pardillo[2], Cristina Cachero[2], and Ambrosio Toval[1]

[1] Department of Informatics and Systems
University of Murcia (Spain)
{fmolina, atoval}@um.es
[2] Department of Informatics and Systems
University of Alicante (Spain)
{jesuspv, ccachero}@dlsi.ua.es

Abstract. It is a proved fact that the appropriate management of requirements is one of the most influential factors in the success of software development projects. With the advent of the Model Driven Engineering development paradigm, the need for formal gathering mechanisms, which provide the necessary degree of non-ambiguity and detail, has led to the proposal of a myriad of requirements metamodels. However, a great disparity exists, both in the concepts/relationships and in the underlying semantics involved in each metamodel. Moreover, most existing proposals lack backward (e.g., alignment with business goals) or forward (e.g., connection with validation methods) traceability. In view of this situation, this paper proposes a 'measurable requirements metamodel' that offers support to the elicitation of measurable requirements. This support is based on the explicit connection of goals, requirements and measures, thus fostering a goal-driven measurable Requirements Engineering perspective. Additionally, since it is well known that metamodels only reflect the abstract syntax of the modelling language, the proposed metamodel also includes a notation (concrete syntax) which, for reasons of understandability, is based on the Goal-oriented Requirements Language notation. This notation is supported by a UML profile which facilitates its adoption by Requirements Engineering analysts in the context of any UML-based software engineering process. In order to support this proposal, an Eclipse tool has been developed. This tool permits the integration of measurable requirements as a driving force in the context of a given Model Driven Engineering development process.

1 Introduction

Goal orientation is a recognized paradigm through which to elaborate, structure and analyse software requirements [1,2,3]. Goals are prescriptive statements of intent whose satisfaction requires the cooperation of agents in the software and its environment. They may refer to functional or non functional concerns and range from high-level strategic concerns to those which are low-level and technical. Unlike other Requirements Engineering (RE) paradigms, goal orientation and its associated Goal Modelling Languages (GML) and elaboration methods, such as KAOS [4] or NFR [5], promote the use of visual models rather than textual descriptions to reason about the system. For the construction of these models, it is common to distinguish between early requirements (organizational goals) and late requirements (system properties and service capabilities), depending upon whether the focus is on the business goals or on the requirements of the system-to-be [6].

On the other hand, software metrics are known to be of paramount importance in increasing the level of objectivity with which software products are evaluated and compared [7]. They are not only a cornerstone piece to help assess and institutionalize software process improvement practices (*e.g.*, ISO/IEC 15504 [8] or CMMI [9]), but they are also useful instruments to validate requirements, usually in the context of quality scenarios [10].

* This work has been partially supported by the projects DEDALO (TIN2006-15175-C05-03) and PANGEA (TIN2009-13718-C02-02) from the Spanish Ministry of Science and Technology, MELISA-GREIS (PAC08-0142-335), ESPIA (TIN2007-67078), QUASIMODO (PAC08-0157-0668) and DEMETER (GVPRE/2008/063). Fernando Molina and Jesús Pardillo are partially funded by the Fundación Séneca (Región de Murcia) and the FPU grant AP2006-00332, respectively.

Finally, but of no less importance, the Model Driven Engineering (MDE) paradigm [11] is increasingly drawing practitioners' attention as an emerging approach for cost-effective, reliable and rapid application development [12]. MDE advocates visual models, which are formally described by means of metamodels, as key engineering artefacts throughout the engineering lifecycle. A metamodel offers numerous advantages [13]. First, the metamodel defines both the elements that participate in the requirements management process and the relationships between them in an unambiguous manner. Moreover, it offers a formal basis upon which tools for the management of the metamodel elements and the definition of transformation rules from requirements to other elements can be constructed. MDE technologies make use of these advantages and combine (1) domain specific languages, whose type systems formalize the application structure, behaviour and requirements within particular domains, and (2) transformation engines and compilers that analyse models and synthesize various types of artefacts (*e.g.*,models with a lower abstraction level or source code).

The integration of goal-driven requirements models, software measures and MDE processes with automation capabilities provides clear benefits:

- **Goal Oriented RE (GORE)+Measures:** the inclusion of well-defined and unambiguous measures in association with requirements increases the accuracy and repeatability of the requirements validation process.
- **Measures+MDE:** the definition of measures as models that can be integrated into MDE approaches permits the measurement task to be automated, thus speeding up the requirements validation process. More importantly, MDE approaches provide early models on which to carry out measurements, thus permitting an early detection and correction of problems, which drastically decreases the project costs [14].
- **MDE+GORE:** in the context of MDE approaches, goal-driven requirements metamodels and models provide a widely used vocabulary and notation for, respectively, visually representing both early and late requirements models that are aligned with stakeholders' objectives.

However, to the best of our knowledge, the integration of these three elements has not yet been considered. Our work aims to fill this gap by providing an integration of goals, late requirements and measures through the definition of the metamodels necessary to support measurable requirements. Specifically, and in addition to this integration, the main contributions of this paper are:

- The definiton of a late requirements metamodel that connects late requirements with goals, thus fostering the traceability between these two concepts. This goal-oriented late requirements metamodel is called **ULR** (Unified Late Requirements metamodel), and was designed after a systematic review of the existing approaches for requirements metamodelling, contributing to the completeness of our approach.
- The combination of the new ULR with an adapted measurement metamodel, called SMM– ('Software Measurement Metamodel'), to create a new 'Measurable Requirements Metamodel' (MRM). The MRM metamodel provides users with the abstract syntax (semantics) for the specification of measurable requirements and converts our proposal, to the best of our knowledge, into the first metamodel to provide support for goal-driven measurable requirements.
- The definition of a UML profile [15] that facilitates the inclusion of these new goal-oriented measurable requirements models in existing MDE approaches, which in turn facilitates the early validation of requirements over the MDE modelling artefacts. This profile provides the concrete syntax (notation) that serves to visually represent the metamodel concepts. The use of a profiling mechanism facilitates its adoption by practitioners, who, according to a recent study, are already accustomed to using UML models [16]. Furthermore, this profile is based on GRL [17], a standard goal modelling language based on **i***, which facilitates its adoption by the RE community, who will not need to learn 'yet another notation'. The profile simultaneously allows the use of the well-known **i*** notation to express a concern that is not present in **i***, which is that of measurable requirements.

The remainder of this paper is organized as follows. In Section 2, the background for this work is provided. In Section 3, the summary of a systematic review of the requirements metamodelling field and the ULR metamodel are presented. Section 4 outlines the SMM– metamodel and explains how both metamodels have been merged into the MRM. Section 5 provides an in depth explanation of the UML profile that has been united with MRM. In order to further clarify the approach, Section 6 presents an example that illustrates the application of the proposed modelling framework. This framework is supported by a CASE tool which

is presented in Section 7. Finally, in Section 8, the main conclusions are drawn and further lines of research are outlined.

2 Background

The tasks related to requirements are among those which are most influential in the success of a software development project. Just as an inadequate management of requirements has been the proven cause of the failure of numerous software projects, their appropriate management helps to improve the quality of the software products and the productivity of their development processes [18]. Throughout its history, therefore, the RE discipline has developed a wide number of approaches that stem from different paradigms, such as goal-driven (GDRE) [3], aspect-driven [19], viewpoints-driven [20] paradigms, or those based on requirements templates [21], to name but a few. Of these, GDRE approaches have been gaining momentum for a number of reasons, including their capability to reason about requirements at different levels of abstraction. As was mentioned previously, they support the definition of early requirements models that represent an organizational domain and serve to analyse it in terms of the business actors and their intentions. They also serve to define late requirements models, in which the software system is represented within its operational environment along with its expected functionality and relevant characteristics [6].

These goal-driven requirements models are supplanting traditional modelling artefacts such as UML use cases or requirements templates [22]. However, this goal-driven perspective has not yet reached the MDE community. Within this paradigm, where models need to act as blueprints of the system, requirements models have traditionally been limited to extended UML use-case diagrams, in which only functional (late) requirements are considered. In these proposals, the limited scope (functionality) and the high level at which such requirements have been described have caused them to influence the MDE transformation process in a solely tangential manner. This has also motivated the definition of requirements metamodels. However, the existing proposals for this aim have certain problems. For example, in some approaches such as [23,24], their high coupling with specific domains have caused their usefulness as general purpose communication artefacts to be hampered. A further important concern is that the traceability between the detailed requirements and the underlying early requirements (business goals) is missing in all these approaches. One noteworthy exception to this is the proposal of [25], in which a goal-oriented requirements metamodel that describes how the resulting models can be used to partially derive and validate the quality of other models is presented. However, the proposal is specific to the Web domain, and the usability validation process is not automatable.

On the other hand, workflows and notations to construct organizational (early) requirements models are an extensive area of work in GDRE, usually included in general GDRE methodologies such as TROPOS [26], KAOS [4], GBRAM [27], NFR [5] or GARE [17], to name but a few. Various Goal Modelling Languages (GML) associated with these proposals have been proposed. Of these, GRL [17], which is associated with the GARE approach, stands out due to its inclusion in the User Requirements Notation (URN) standard [28]. Most proposals also include support for late requirements views, and even implementation technique catalogues that help to achieve certain requirements (see, *e.g.*, NFR). The means to address the transformation between organizational models and late requirements models varies. In [6] it is achieved through the use of patterns. TROPOS defines a bridge between organizational stakeholders and goals, and the candidate software architectures supporting the requirements. In KAOS, the change from early to late requirements is achieved through the operationalization of goals, in which responsibilities for their fulfilment are assigned to agents. In GBRAM, the definition of late requirements from organizational goals is based on scenario techniques. However, contrary to our approach, the complexity of the resulting requirements models in all the aforementioned approaches is unsustainable for most MDE processes, which are usually adopted in cases where baselines need to be generated very quickly, and requirements techniques therefore need to be extremely lightweight, intuitive, and usable by software analysts and project stakeholders, who often have little or no technical background [25].

We claim that, in order to palliate these shortcomings, MDE approaches require the inclusion of a requirements metamodel that borrows the explicit consideration of the complementary GDRE concerns (early requirements and late requirements) as part of any requirements metamodel included in an MDE process. This notwithstanding, this metamodel should be simple, signifying that only the core concepts should be present. We claim that this simplicity means to centring them on late requirements, which are those that make a direct impact on the modelling artefacts, and simply provide a link to early requirements, which

can be worked on outside the MDE process. Furthermore, we are of the opinion that it is of the uttermost importance that this metamodel additionally supports the link between late requirements and measures, in order to support their early validation over the MDE modelling artefacts. Furthermore, in order to define a sound late requirements metamodel that can be derived from GDRE approaches and is simultaneously simple enough to be adopted by MDE practitioners, we claim that other concerns need to be addressed: requirements metamodels for MDE should strive towards reuse, and include traceability among early and late requirements. However, detailed reasoning mechanisms to solve aspects such as conflicting requirements, given their complexity, should be left aside, assuming that they can be solved in a previous step, namely during the transformation from the organizational model into this late requirements model.

Our approach covers all these aspects. Furthermore, and given the existing disparity with regard to the semantics of the RE concepts included in the different GDRE metamodels [29], our proposal is based on a systematic review in which the most common acceptations for concepts have been incorporated. Finally, but of no less importance, our approach establishes the role that this late requirements metamodel should play in the MDE transformation process, that is, to serve to drive the automatic validation of requirements over other MDE artefacts.

3 The Unified Late Requirements Metamodel

3.1 Introduction

As we have previously mentioned, the different existing requirements metamodelling proposals present a great disparity in their level of abstraction, number of concerns addressed, and semantics. In view of this situation, and given the fact that late requirements metamodels are those that provide the most useful information from an MDE perspective, the following section presents a summary of a systematic review carried out to identify the most relevant late requirements metamodels (be they GDRE proposals or otherwise). This review has permitted us to identify their core concepts.

3.2 A summary of a requirements metamodels systematic review

The aforementioned literature review was carried out by following the guidelines proposed by [30], which establish a set of stages with this aim, which are: (i) question formularization, to clearly define the research objectives; (ii) source selection, in which the sources for the searches will be selected; (iii) studies selection process, which describes the process and the criteria for studies selection and evaluation; (iv) information extraction, which defines the extraction criteria and (v) extraction execution, which presents the data resulting from the selected studies.

In our case, the main aim of the review was to identify what the existing proposals for requirements metamodelling were, and what concepts they covered. A set of sources (namely IEEE Computer Society, ACM Digital Library, Google Scholar, Citeseer and ScienceDirect) that contain the work published in journals and conferences of recognized quality within the research community was analysed. These sources were subjected to various search string criteria. That which retrieved the highest number of useful results was: (*"requirements" AND ("metamodeling" OR "metamodelling" OR "metamodel")) OR ("metamodel of requirements"*). As a result, six proposals for requirements metamodelling were identified. The result of their comparison can be seen in Table 1 and more details of the systematic review can be found in [31].

As Table 1 shows, the number and type of concepts included in the different metamodels vary greatly, the **requirement** concept being the only one which is universally present. Unfortunately, we have been unable to assess whether the underlying semantics are the same in all the approaches, since none of them provides formal definitions of the included concepts. Furthermore, there is no consensus with regard to whether or not requirements should be subcategorized, the most common classification - when present - being that of functional versus non-functional.

From the six metamodels identified, three of them [25,32,33] (that is, 50%) include as a cornerstone the **goal** concept, which permits late requirements models to be connected with organizational models elaborated in previous steps in any GDRE process. However, as the reader may have already observed in Table 1, there is little else in common among these goal-aware metamodels.

	Bolchini [25]	COMET [32]	Fernández [33]	Goknil [34]	REMM [35]	SME [36]
Additional Desc./Use case		✓	✓	✓		
Catalogue					✓	
Functional Requirement		✓	✓			✓
Glossary				✓	✓	
Goal	✓	✓	✓			
Non Functional Requirement		✓	✓			
Requirement	✓	✓	✓	✓	✓	✓
Scenario		✓	✓			
Source					✓	
Stakeholder	✓	✓	✓(Actor)	✓	✓	✓
Term			✓	✓	✓	
Test Case				✓	✓	✓
Trace				✓	✓	

Table 1. Late Requirements Metamodels' concepts comparison

Another concern that appears in some metamodels [35] is that of the reuse of requirements, which is tackled by including a **catalogue** concept. This concept serves to gather a set of requirements extracted from one or more **sources** (*i.e.*, a law, an organizational policy, a particular domain, a set of guidelines,...) which can be reused in all the **projects** in which these sources are applicable [37].

Yet another concern is that of vocabulary disambiguation. Since requirements are usually described by using natural language, which may be imprecise and ambiguous, some metamodels [33,34,35] have considered the possibility of describing requirements through the use of a set of well-defined **terms** collected in a **glossary**.

Also detected in various metamodels [32,33,34] is the concept **additional description**. On many occasions, a simple textual description is not sufficient to make a requirement understandable. Some metamodels therefore introduce this concept to provide more detailed information about the requirement, using text or some kind of artefact which is appropriate for this aim, such as UML use cases.

Two [34,35] out of the six metamodels include a tracing concern, which is understood to be a relationship that exists among two or more requirements and which is reflected in the trace concept. These traces can be classified according to the kind of relationship: refinement, conflict, etc.

It is worth noting that some proposals [34,36,35] have taken into consideration the requirements validation concern, that is, the need to check that the system or software implemented fulfils a predefined requirement. In order to address this concern, such metamodels include the concept of a test case. However, no guidelines are provided regarding how such a validation should be defined in order for it to be repeatable, systematic and, if possible, automated.

Finally, but of no less importance, our comparison reveals that only some of the metamodels (*e.g.*,[35]) offer some kind of automatic support to deal with the concepts proposed by the metamodel, and only a subset of them offers a concrete syntax (notation) to deal with the concepts that make up the metamodel.

During the comparison we discovered some concepts which are particular to individual metamodels, such as tasks, features, viewpoints or design constraints. These concepts usually relate to the specific domain for which the metamodel was designed, and they have been left out of Table 1 for reasons of space. Based on this systematic review, the following subsection shows our proposal for a late requirements metamodel for MDE.

3.3 ULR: a Unified Late Requirements metamodel for MDE

Given the lack of agreement on the core concepts for a late requirements metamodel, we have based our proposal on the following set of concerns and constraints. On the one hand, the set of concerns was defined

by using the best practices that were detected during our review of late requirements metamodels, and included: (1) support for requirements reuse, (2) support for backward and forward traceability, (3) support for validation, and (4) support for both functional and non-functional requirements. On the other hand, the set of constraints was determined both based on the late-nature of the model (signifying that it had to be focused on the requirements of the system-to-be, regardless of the means by which those requirements were obtained) and on the need for it to be unambiguous and specific enough to allow it be integrated with MDE practices and agile methodologies. It also had to be kept simple, in order to facilitate both its quick instantiation and its easy manipulation in MDE transformations.

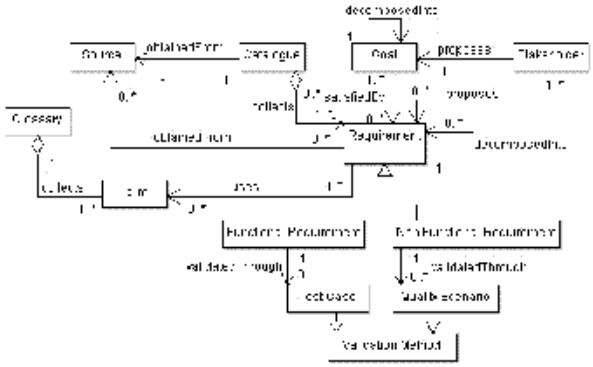

Fig. 1. ULR: The Unified Late Requirements metamodel

Figure 1 shows our ULR proposal. As occurred with all the late requirements metamodels studied, the key element in ULR is that of `requirement` which can be described by using a set of attributes (hidden in the figure) such as an `identifier`, its `type`, its `priority`, and its textual `description`. Requirements can be classified as either `functional requirements` (what the system must do) or `non-functional requirements` (how the system must do it). This classification is useful because, while functional requirements usually rely on `test cases` for their validation, non-functional requirements are related to `quality scenarios` that quantify and contextualize a given quality concern [10]. More complex requirements classifications have been avoided for the sake of simplicity. Requirements which are organised into tree-like structures (see, *e.g.*, quality models such as the ISO 9126 [38]), along with relationships between functional and non functional requirements, are captured by means of the unary relationship `decomposedinto`, which is defined over the requirement concept. On occasions, `catalogues` originating from certain `sources` are useful resources from which to pick up requirements, thus speeding up the metamodel instantiation process.

Due to the need for traceability with business objectives, which are gathered together in organizational models, each requirement is connected through the binary relationship `satisfiedBy` to the set of `goals` it contributes to satisfying. Each goal can be `decomposedInto` other goals, and captures a high-level objective of one or more `stakeholders` that propose it.

Moreover, in order to, as far as possible, avoid the ambiguity inherent in natural language, and to help developers understand the requirements (often expressed in terms that are only familiar to domain experts), the description of requirements may be accompanied by `terms` included in a `glossary`.

Now that this metamodel has been defined, the following section explains how it can be connected with a measurement metamodel in order to offer support to the concept of 'measurable requirement', and thus promote the forward traceability of requirements.

4 The Measurable Requirements Metamodel (MRM)

The explicit definition of both organizational and late requirements models and GDRE methodologies, although necessary, may not be sufficient to guarantee the quality of the resulting application [39]. Indeed, the explicit alignment of business goals with software requirements could lead to good products. However, many other factors may influence the quality of the products (*e.g.*, human decisions). It is thus necessary to complete methodologies with techniques for product quality assessment. This quality assessment is considered to be so critical that specific concepts for its management have been included in a proposal for a core (and thus minimal) requirements ontology [29]. This ontology clearly states that any 'functional goal' and any 'quality constraint' should be verifiable, in the respect that the software engineer should be able to check, at any time, whether the system satisfies them. This same need for validation instruments is stressed in [40]. While functional requirements are usually easily validated (in the sense that the software engineer can determine whether what is functionally required is indeed delivered by the system), non-functional requirements require a well defined, objective, quality space that is shared among the stakeholders. If such objective quality space is missing, we risk biased, subjective quality assessment [29]. In order to achieve this objectivity, requirements need to be quantifiable. One widespread method for quantifying non-functional requirements consists of defining associated measures which make them measurable [41]. This practice has been included in many software development processes (see, *e.g.*, [10]), in which it has been proved to contribute to the success of projects. In order for these measures to be supported by an MDE process, it is necessary for its definition to be supported by metamodels. However, this aspect has, to the best of our knowledge, been systematically neglected in late requirement metamodels (see Section 3).

In order to overcome this shortcoming, and to pave the way for an automation of the whole measurement process, we need a metamodel that supports the systematic definition of measures, and a link between this metamodel and the requirements they are bound to quantify [42]. Our proposal to meet these needs is denominated as SMM− (see Figure 2), which permits measures to be defined in a systematic and unambiguous manner. Its name is derived from the fact that its construction is a simplification of the well-known Software Measurement Metamodel (SMM) [43], which was created based on a Software Measure Ontology (SMO). This ontology provides a certain degree of harmonization to the different software measurement standards and research proposals, along with a common conceptualisation of software measurement, in which objects, concepts, entities and their relationships are explicitly represented in an unambiguous and explicit manner. The rationale behind the necessity for this simplified view over the original SMM metamodel is twofold; on the one hand, some of the concepts in the original SMM were defined at different levels of abstraction, which were not suitable for our purpose: e.g. in the original SMM, `Entity` is an instance of `Entity Class`. Therefore, only `Entity Class` remains in our metamodel. On the other hand, some SMM concepts simply overlapped with the concepts in ULR: the SMM `Information Need` is, from our point of view, an oversimplification of a whole requirements model, such as our ULR model. Another overlapping occurs with the SMM `quality model` and `measurable concept` terms. In our approach, the `quality model` is implicitly defined by the set of non-functional requirements included in the ULR model. These requirements need an associated `Validation Method` which can be linked with a measure. In our approach the SMM `Measurable Concept` has therefore been decomposed into a ULR requirement that is associated with an SMM− measure.

In the SMM− metamodel we can observe that everything is organized around the `measure` concept. Measures can be subdivided into `base measures` (directly calculated by means of a measurement method), `derived measures` (based on one or more base measures, and calculated by means of a measurement function) or `indicators`. Indicators, calculated by means of `analysis models`, can be made up of both base and derived measures, and have certain `decision criteria` associated. Generally speaking, the decision criteria translates measurement results into answers to a certain information need. In other words, in our framework, indicators serve to validate non-functional requirements. In order to fully specify a measure, the `scale`, the `unit of measurement` and the `type of scale` must be provided. Moreover, measures are applied over a certain `entity type`, which, in our case, corresponds with any kind of model included in the MDE process. Models have certain `attributes` (namely size, complexity, length, coupling and cohesion [44]), and measures can be formally related to any of these attributes.

In order to link the ULR and the SMM− (see Figs. 1 and 2), it is necessary to provide a relationship between (i) the `Validation Method` metaclass of ULR on the one hand, and (ii) certain `Measures` of SMM− on the other. Since `Indicator` is the only measure type that provides a means to translate the measurement

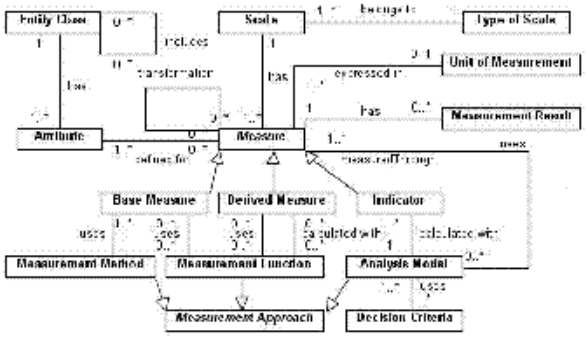

Fig. 2. SMM−: A simplified Software Measurement Metamodel

result into a validation criteria (through the decision criteria metaclass), we have merged both metamodels together with the addition of a `Measured Through` link between the `Validation Method` and the `Indicator` metaclasses. The metamodel resulting from this integration (see the central part of Figure 3) is what is actually called MRM.

It is important to note how the resulting MRM supersedes not only ULR, which can now define measurable requirements, but also the original SMM itself which, with the linking to RE concepts, can now soundly express not only the measures but also the context and purpose of the measures in a systematic manner. In MRM this context not only involves specific goals or specific requirements, but also specific quality scenarios with specific conditions under which the measures are to be applied. All this information is of paramount importance in the definition of measures, as they affect the acceptability of the measure result. For example, the application of a performance measure with a system load of 10 simultaneous users (quality scenario 1) is different to the application of the same measure with 5000 simultaneous users (quality scenario 2).

In order to summarise this, Figure 3 shows our metamodelling architecture. In this figure we can observe how the MRM imports the ULR and the SMM− metamodels. The figure also includes a `MeasuredThrough` element that provides the hook between these two metamodels, by means of a link between the Validation Method concept (defined as part of the ULR) and the Indicator concept (defined as part of the SMM−).

Although the resulting MRM metamodel can be used by directly instantiating the metaclasses to specify measurable requirements, experience has demonstrated that object models do not scale well [45]. The following section therefore presents a visual notation that provides MRM with a concrete syntax and helps modellers to deal with the MRM instantiation.

5 A Visual Notation for MRM: the MRM Profile

If we look back at Figure 3, we can observe that the metamodels shown in Figures 1 and 2 have been represented by means of MOF (Meta-Object Facility) [46], a widely used standard for metamodelling that provides useful primitive types such as classes, properties, and associations. Despite the fact that MOF has been recommended by the Object Management Group (OMG [47]) for general metamodelling in MDE projects, its lack of visual notation makes it an unsuitable candidate for practical modelling. In fact, just as a proper requirements metamodel is necessary to improve software analysts' communicational capabilities, the provision of a proper visualisation for the corresponding requirements model is necessary to improve its usability as an analysis artefact [48]. We therefore advocate not only formally specifying requirements based on metamodelling, but also diagramming these models in a manner that will improve their cognitive effectiveness.

To achieve its maximum potential, MRM visual models should be aware of the principles that empower the human's visual and cognitive capabilities involved in understanding the models [45,49]. The raw definition of object diagrams that represent metaclass instances (*classes*, *relationships* and *properties*) with a neutral

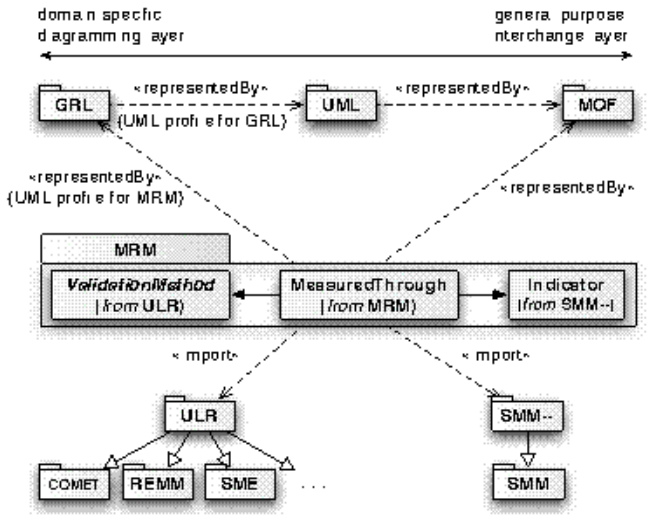

Fig. 3. MRM modelling architecture for MDE GDRE

notation (`InstanceSpecification`, `Link`, and `Slot`, respectively, see Figure 9(a)), is not, therefore, sufficient [50]. An additional rendering layer for the MRM metamodel that provides it with a notation that takes into account recent research on cognitive complexity management [45] is thus necessary.

For the definition of this notation, and given the many syntactic variations in the existing GMLs [51], we have opted to base our modelling framework on the User Requirements Notation (URN) standard [52], which adopts one of the aforementioned proposals for GMLs, denominated as GRL, for the definition of business goals, non-functional requirements, alternatives and rationales. However, since GRL does not provide the concepts needed to deal with measurable requirements, the MRM modelling framework includes a notational extension of GRL. For this extension, and given that the notation should preferably be modular [53], we have decided to use the well-known profiling mechanism provided by UML. The result is an MRM profile, which is based on a pre-existing GRL profile, and which extends it with measurement concepts.

MRM can therefore be seen as a two-layered modelling framework (see Fig. 3): a ***diagramming layer*** (based on a UML profile for GRL) and an ***interchange layer*** (based on MOF, *i.e.*, raw metamodelling). The former (see the top left-hand side of Figure 3) deals with an extended version of GRL diagrams (familiar to RE analysts). The supporting layer (on the top right-hand side of Figure 3) is an MOF version of MRM that provides a clear, formal metamodel basis. The transition from the UML profile for GRL to MOF is performed ***via model-to-model*** transformations by following the MDE technologies [11]. Whilst the main concern of the ***diagramming layer*** is to visually manipulate requirements models, the ***interchange layer*** stores the resulting metamodel in a platform-independent format. This architecture permits analysts to use a UML tool for diagramming while still reusing the GRL notation. Furthermore, they can benefit from a model validation, storage and later transformation which are carried out in a vendor-independent manner, thanks to the interchange of raw MRM models.

5.1 Overview of the mapping of GRL into UML

When preparing any language in order to allocate another language (in our case GRL in order to allocate MRM), the host language must include some sort of extension points to enable it to be adapted to its notation. The required mechanism can be as simple as modelling element annotation capabilities in terms of notes,

tags, or description-like attributes. The only practical condition is that the renderer of the host diagrams must be aware of these annotations in order to show them correctly in the diagram. For instance, GRL includes description attributes in its metaclasses that could be tagged to accommodate the MRM concepts. The drawback of this approach is that the new concepts would lack visual icons. As another alternative, GRL tools provide notes that can be attached to modelling elements to show the extension. However, the use of notes results in overloaded diagrams which are not easy to understand. The conclusion of this analysis is that GRL is not well suited to hosting the MRM.

All these problems can be easily solved if we use the UML language. While GRL was not originally conceived for allocating extensions that are undertaken by using ad-hoc mechanisms, UML supports the definition of UML profiles, a formal extension mechanism which is both semantic and notational, and which elegantly permits UML to host other languages. UML profiles are an intuitive, easy-to-use mechanism through which metamodels can be enriched in unforeseen ways. By profiling UML, we can formally overcome the extensibility limitations of GRL. In addition, the use of UML as a scaffolding enables us to employ MRM models as an additional artefact or perspective in the context of a development process based on UML modelling tools.

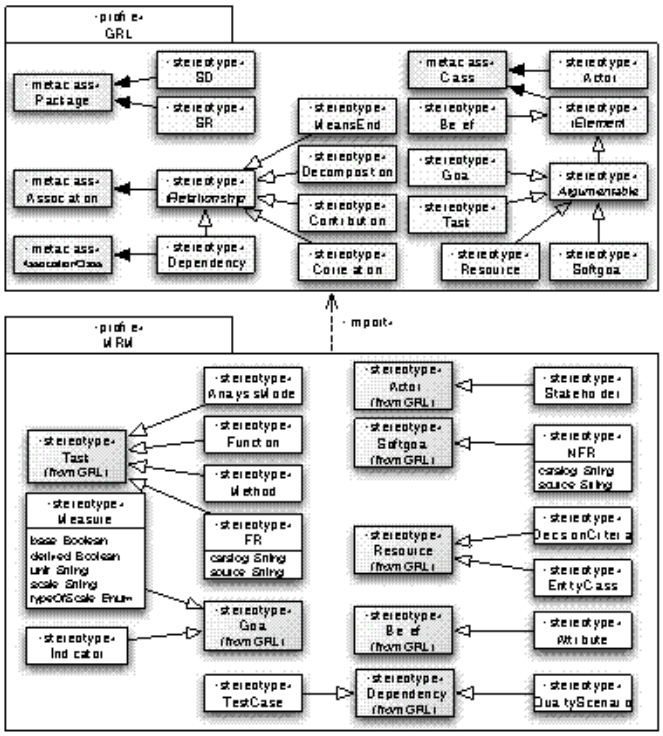

Fig. 4. UML profiling architecture for diagramming MRM models

Therefore, provided that we use a UML profile as a starting point for GRL, UML provides all the necessary mechanisms to define the extension to support MRM. Such a profile has already been proposed in Mazón *et al.* [54] (see top package in Figure 4). The GRL profile is based on the metaclasses of the UML metamodel

which are most commonly used to define profiles, namely, *Package*, *Association*, *AssociationClass* and *Class*. In Figure 4 we can observe how they have been adapted to deal with the GRL semantics: there are stereotypes for the strategic dependencies diagram (*SD*), the strategic rationale (*SR*), intentional actors (*iActor*), the different types of intentional relationships (*MeansEnd*, *Decomposition*, *Contribution*, *Correlation*, and *iDependency*) and the set of intentional elements (*Belief*, *Goal*, *Task*, *Resource*, and *Softgoal*). It is important to note how this profile provides modellers with both the semantics of GRL elements and their notation in UML.

5.2 A UML profile for rendering MRM

The GRL profile provides the basis for the definition of the new MRM profile, by means of the UML specialization mechanism. Table 2 shows the new MRM stereotypes, defined based on the names of the MRM metaclasses together with the intentional element of the GRL profile that serves as a basis for the extension (see bottom package in Figure 4). This specialization contrasts with the UML *Extension* relationship that was used between metaclasses and stereotypes in the original GRL profile (see top package in Figure 4). The reason for this is that, while in the GRL profile the elements extended were UML metaclasses, for the MRM profile we are extending stereotypes, in order to adapt an existing profile to accommodate the new MRM elements. This adaptation is achieved through the definition of inheritance hierarchies between UML classifiers, by means of *Generalization* relationships.

Metaclasses Mapping. The MRM extension presented in Figure 4 is derived from the notational mapping presented in Table 2. In this Table, the selection of the notation has been carried out by following a sound set of diagramming principles that foster cognitive effectiveness [45], since it is well known that this effectiveness is primarily determined by the perceptual characteristics of the diagram. As stated in [55], even slight changes in graphical representation may make a dramatic impact on understanding and problem solving performance. These principles are, namely:

Perceptual discrimination: in our proposed notation, different shapes are associated with different concepts, to foster their recognition (compare the diagrams depicted in Figures 9(a) and (b)).

Perceptual configuration: our notation proposal is based on icons; icons make diagrams more visually appealing, speed up recognition and recall, and improve intelligibility to novice users. Furthermore, the inclusion of some objects inside others (by means of actor boundaries) creates perceptual grouping, which expands the number of elements that can be shown in each diagram without exceeding cognitive limits. This mechanism is complemented by the use of packages, which are provided by the UML scaffolding and contribute to improving the end user's understanding and the verification accuracy by more than 50% [53].

Perceptual precedence: in our notation proposal not all the metamodel concepts have an associated icon. Given the fact that the human working memory is very limited, the mind needs to establish the order in which elements are dealt with; the definition of some concepts by means of tags rather than stereotypes helps the user to choose which elements to pay attention to first. For example, if we look at the *Measure* stereotype, it maps two MRM metaclasses (*Base Measure* and *Derived Measure*); the represented elements are discriminated by means of tagged values associated with the stereotype. Other metaclasses have been similarly notationally translated into tag definitions (see Table 3). Moreover, the *Glossary* and *Term* MRM metaclasses have not been provided with a modelling mechanism, since their textual nature makes them poor candidates for diagrams.

Perceptual directness (notational knowledge): the fact that our proposed notation uses GRL icons (shared by most i^* proposals), and even reuses GRL concepts (and not only notation) when possible (see, *e.g.*, the goal element), facilitates its recognition by analysts who are familiar with the GRL notation. What is more, the use of stereotypes provides a standardized element labelling. This has been shown to be essential in diminishing the cognitive overload that hampers the understanding of diagrams.

Metarelationships Mapping. As the reader may have already noticed, in our profile only inter-actor relationships have been stereotyped. The task dependency metarelationship represents test cases that connect functional requirements (represented as tasks) with indicators (represented as goals). The resource dependency metarelationship is used to represent either (i) quality scenarios, which link non-functional requirements (represented as softgoals) and indicators (represented as goals) or (ii) measurement results, which link requirements with indicators.

Table 2. Representing MRM metaclasses by means of GRL

GRL Element	Notation	Equivalence in MRM
Actor	◯	Stakeholder
Goal	⬭	Goal, Indicator, Base Measure, Derived Measure
Softgoal	⬭	Non Functional Requirement
Task	⬡	Functional Requirement, Analysis Model, Measurement Function, Measurement Method
Resource	▭	Attribute, Entity Class, Decision Criteria
Task Dependency	-◯-	TestCase
Resource Dependency	-▭-	Quality Scenario, Measurement Result

With regard to the relationships between the MRM metaclasses, the designed profile does not contain stereotypes for adapting the GRL relationship notation (see Figure 4). This is due to the fact that every MRM metarelationship between each pair of MRM metaclasses is mapped into just one single intentional relationship in GRL. Therefore, by attending the source and end metaclasses, both modellers and model-management tools are able to trace the source MRM metarelationship. In order to illustrate this, Fig. 5 shows the mapping of some SMM– metarelationships. This Figure allows us to observe how indicators (mapped into goals) and analysis models (tasks) are related through a *means-end* relationship. The same relationship appears between derived measures (goals) and their corresponding measurement functions (tasks) and base measures (goals) and their corresponding measurement methods (tasks). A further relevant intentional relationship is that of task ***decomposition*** which appears between analysis models (tasks) and their related measures (goals) or decision criteria (resources), and also between measurement functions (tasks) and the associated measures (goals), or between measurement methods (tasks) and the associated entity classes (resources). Finally, a ***contribution*** relationship provides the attribute (belief) that permits the connection between measurement methods (tasks) and entity classes (resources).

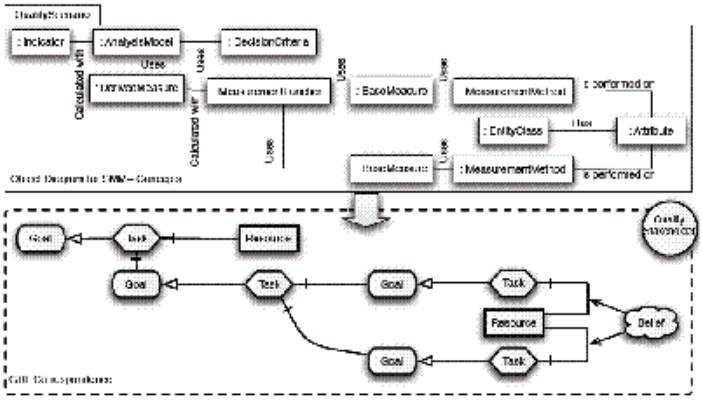

Fig. 5. GRL notation for MRM measurement concepts

Metaproperties Mapping. The metaproperties involved in MRM are shown in Table 3. These metaproperties are mapped into the corresponding tag definitions (*i.e.*, metaproperties profiling implementation), which are associated with the stereotypes shown in Figure 4.

Table 3. Tag definitions involved in the MRM profile

Stereotype	Tag	Data Type	Semantics From MRM
for all	id	String	*core facility*
	description	String	*core facility*
FR, NFR	catalog	String	Catalog
	source	String	Source
Measure	base	Boolean	Base Measure
	derived	Boolean	Derived Measure
	unit	String	Unit of Measurement
	scale	String	Scale
	typeOfScale	Enumeration	Type of Scale

Typically, most stereotypes in a UML profile are complemented with several constraints, usually defined in formal languages such as OCL [56], that assure the consistency of the model. For example, in MRM models, the mutual exclusion between the boolean conditions associated with the *base* and *derived* tags must be assured; moreover, the metarelationship mappings presented in Figure 5 must be preserved.

However, the MRM architecture (see Figure 3), and the fact that MRM is targeted towards providing a notational rather than a semantic extension of GRL, makes the definition of these constraints at profile level unnecessary. Instead, MRM models are validated by the MRM semantics constraints in the storing layer, based on MOF. For this validation MRM models need to be transformed into MOF models. We thus avoid OCL expressions that are expensive to codify and prone to failure owing to the lack of formal or automatic methods to guide the constraints mapping involved in the profiled metamodels. What is more, we allow for the reuse of the model checking capabilities independently of particular modelling tools.

5.3 The MRM rendering notation

The MRM rendering notation associated with the MRM profile is presented as follows. Given that the MRM profile is defined in terms of the GRL profile, in order to show the MRM notational changes it is necessary to understand how the GRL profile alters the UML notation.

Fig. 6. Three alternative ways to represent stereotype icons in UML

Metaclasses Visualisation. With regard to the UML profile for GRL [54], but also applicable to any UML profile, there are mainly three alternative notational variations for the iconography introduced by the profile's stereotypes (shown in Figure 6). The first and most classic variation is the simple decoration of the extended metaclass (see the *Class* UML metaclass extended to represent a GRL *Task* in Figure 6). The metaclass therefore retains its original notation. The second variation is another UML standard notation mechanism that can be applied when 'boxes-like' metaclasses (such as *Class*) are rendered without their properties. The third notational variation is used when straightforward visual metaphors are needed to represent the concept, and this implies hiding the underlying UML notation completely. This kind is the most powerful, but it is necessary for the modelling tools to be able to interpret, or at least emulate, the specific non-normative extensions out of default notational extensions provided by UML modelling tools with strict profiling capabilities.

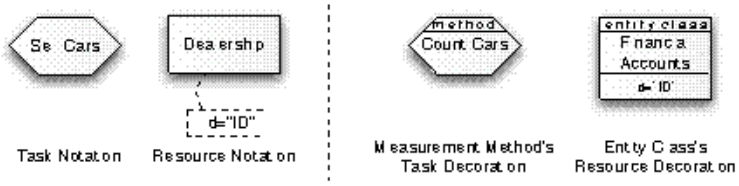

Fig. 7. GRL notation decoration for rendering MRM metaclasses

Metarelationships Visualisation. With regard to the MRM metarelationships supported by the metaclasses mapping (Section 5.2), the stereotyping of intentional relationships is not actually required and the current GRL notation thus works well in this case. Note that, from a practical point of view, the mappings involved between MRM and GRL assure that MRM relationships will be univocally discriminated from the rendering intentional relationship of the GRL profile for MRM.

Metaproperties Visualisation. Figure 7 also shows the decoration of the GRL notation for the MRM metaproperties. As occurs with the metaclasses, metaproperties are contained in a lower compartment of the corresponding node-like rendering (note that the relationships are defined in such a way that they do not have properties), in contrast to the noting mechanism initially provided by GRL (*e.g.*, *Dealership's id* in Fig. 7).

To illustrate this, we shall now present an example of the use of MRM. For the sake of conciseness, in this example we will use the third of the notational alternatives presented in Figure 6.

6 Example

The application example chosen to illustrate our approach deals with a requirements elicitation scenario for a simplified on-line ticket sales system. Both a description of the MRM instantiation and the presentation of the MRM profile notation are discussed.

6.1 Description

For the sake of simplicity, let us assume that the *sales manager* of a certain cinema chain wishes to *increase the sales profit* of the company. Let us also suppose that this goal is to be achieved through two different sub-goals; on one hand the company wishes to *increase the sales net profit* by implementing a Web-based on-line ticket sales system that decreases the costs associated with the sales process. On the other hand, the company also wishes to *increase the number of sales* by obtaining a wider range of customers. The sales manager believes that offering the tickets through the Internet may positively influence both sub-goals, so s/he has taken on a Web ticket sales system development process.

The *Web customer* who will interact with the application has as his/her main functional requirement that of *buying ticket*. This requirement can be decomposed into two functional requirements: *browse films* and *purchase ticket*. In addition, several non-functional requirements have been identified. The *buy ticket* functionality should follow *accessibility* guidelines to allow Web customers with disabilities to access the system. Additionally, the system should provide *information accuracy* while browsing through the films: synopsis, sessions, prices and so forth should be reliable. Also, the application's *learnability* should be high, that is, the application should be simple enough for novice users to learn its operation easily. Finally, the purchase process should be performed while assuring the *security* of the customer data.

6.2 Instantiating the MRM metamodel

If we first check the involved ULR metaclasses (see Figures 1 and 9(a)), we can observe that (i) the sales manager and Web customer both instantiate the **Stakeholder** metaclass, (ii) all goals related to increasing

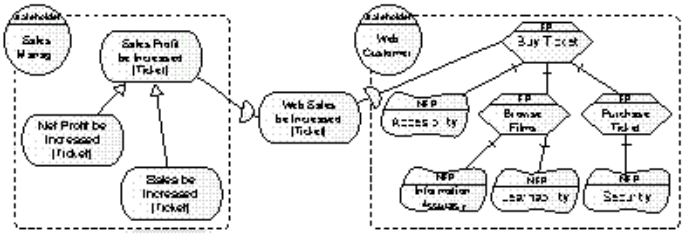

Fig. 8. Ticket Sales System: Stakeholder's goals and requirements

sales profit instantiate the **Goal** metaclass, (iii) buy tickets, browse films, purchase tickets are all **Functional Requirement** occurrences and (iv) accessibility, learnability, security and information accuracy are all **Non Functional Requirement** instances. Also, note how the GRL notation (Fig. 8) improves the understandability of this MRM model rendered as an object diagram (Figure 9(a)).

Let us now focus on the learnability with which the Web customer is able to browse the films we are offering on our site, and let us convert this into a measurable requirement. For this purpose, let us assume that we have defined a quality scenario –instantiated from the **Quality Scenario** metaclass (see Figure 1)– which states that it is necessary to check whether a novice user is able to find the link s/he expected to find based on his/her mental model of the domain.

If we now check the SMM– (see Figure 2), we can observe how the first step in making a requirement measurable is that of defining an **Indicator** occurrence that we have denominated as 'navigability level'. Indicators are made up of measures. In our example, let us imagine there is a single **Derived Measure** instance that contributes to the indicator, which is called 'domain coverage navigation measure' (DCNM). This measure is based on two **Base Measure** occurrences: the 'number of relevant relationships' (NRR) and the 'number of navigated domain relationships' (NNDR). In short, the DCNM calculates how many of the domain relationships that may be relevant in fulfilling the requirement are actually navigation links which are available through the application interface. Let us also imagine that, in this context, the **Decision Criteria** and the **Analysis Model** associated with the indicator establish that a $DCNM > 0.80$ makes the Web design acceptable with regard to its learnability. Both the NRR and the NNDR are measures that can be calculated automatically if the web development process includes conceptual domain and navigation models such as the ones managed in the context of Web Engineering methodologies. Assuming that the system is being deployed with one of these Web Engineering proposals, namely OO-H [57], the DCNM measure can be calculated early in the lifecycle, based on the **Entity Class** 'web conceptual model' (an abstract model that represents an entire Web application). More specifically, the NRR is calculated over the 'domain model' **Entity Class**, while the NNDR is calculated over the 'navigational model' **Entity Class**. Both measures are related to the structural complexity **Attribute** of the associated models. The **Measurement Function** that serves to calculate the DCNM uses the two **Base Measures** in order to calculate the **Measurement Result**. The higher this value is, the more domain relationships are covered in the application interface by means of links.

Figure 9(a) shows the objects diagram that represents the instantiation of MRM for this example. This model can be used in the context of model-driven methodologies to automate the measurement process at every stage of development, and even to automatically evolve the models based on measurement results. Readers interested in the whole approach are referred to[42].

As the reader may note, despite the low complexity of the example, the size of the obtained diagram is quite large, while its understandability is low. The provision of the visual notation improves the conciseness of the diagram (see Figure 9(b)), while simultaneously improving its readability.

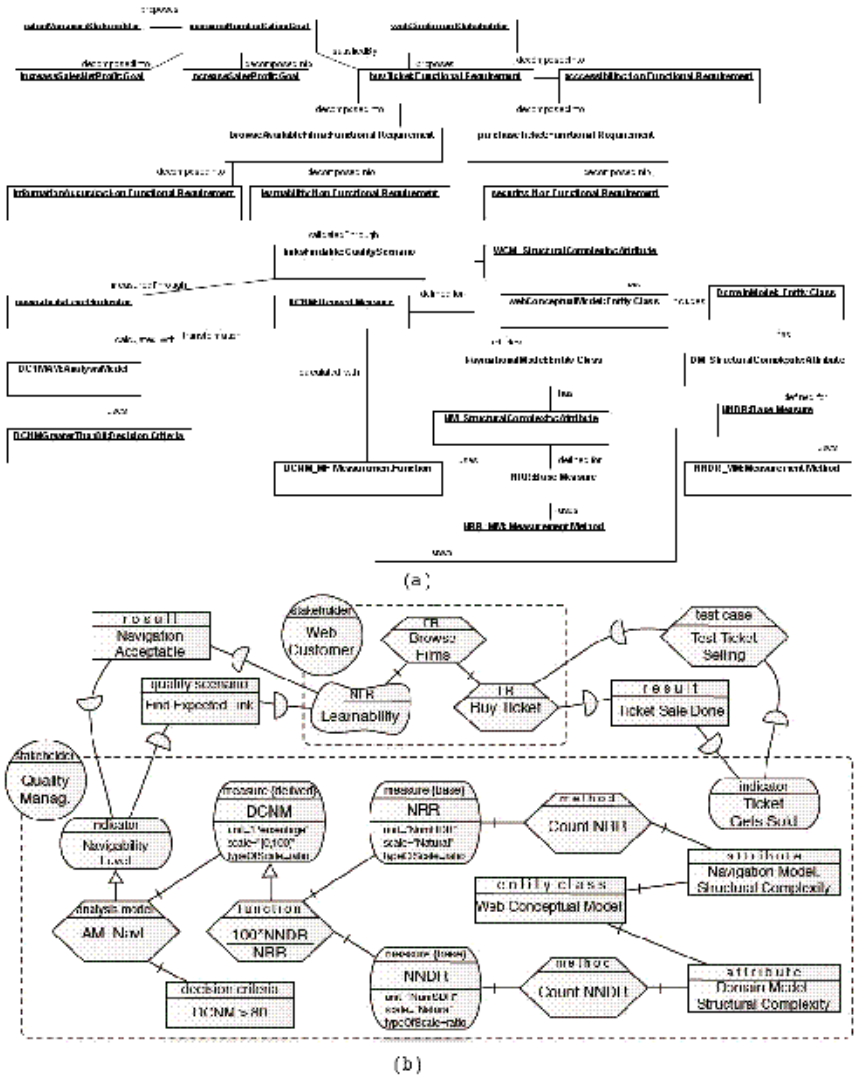

Fig. 9. (a) Object diagram corresponding to the instantiation of MRM for the *Ticket Sales System* (b) Equivalent model using the MRM notation

7 Tool Support for the Management of MRM Models

Given the increasing complexity of the systems being developed, modelling frameworks are currently of little use if they are not supported by CASE tools that help to manage them. Thus, this section presents how the MRM framework has been integrated into an MDE environment.

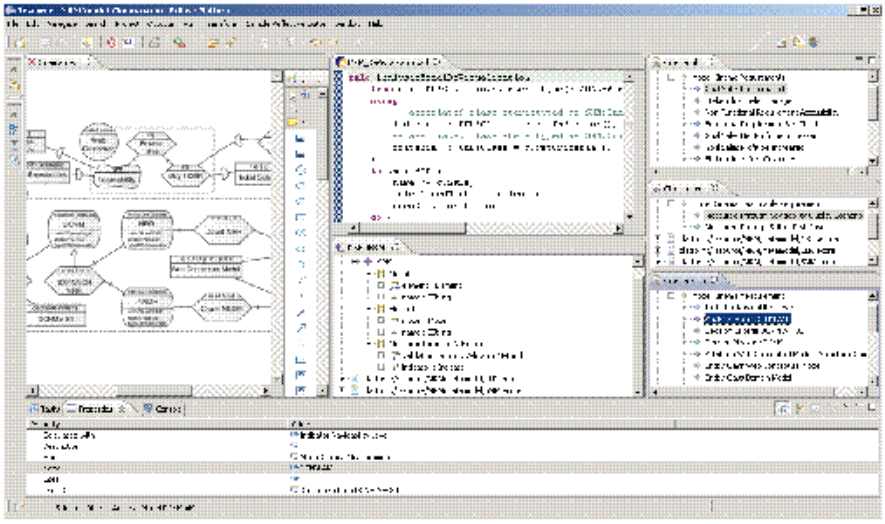

Fig. 10. MRM modelling framework supported by the ECLIPSE development platform

In the case of MRM, and given the fact that it was created with the aim of its being integrated into MDE approaches, our modelling framework is accompanied by a tool that includes a model-to-model transformation architecture for articulating both the visual representation and the storage of MRM models. Certain considerations have been taken into account in the selection of the technological environment in which to implement this tool. First, it was necessary to use an environment that would facilitate the definition of metamodels. Moreover, as we wished various stakeholders (and not only designers) to use MRM, it was necessary to offer them a graphical tool with a usable and comfortable interface that would allow them to create and manipulate models compliant with the metamodel. It was also necessary to consider the capacities offered by the technological environment in order to extend the tool with new functionalities or to integrate it with other pre-existing tools.

Given these premises, the ECLIPSE development platform [58] and, in particular, the ECLIPSE MODELLING FRAMEWORK (EMF) was selected to implement the tool. Both ECLIPSE and EMF have certain suitable features that make them of interest to us in our approach. On the one hand, EMF offers support to deal with MOF. EMF is therefore useful both to describe MRM and for the creation and manipulation of models and metamodels. On the other hand, ECLIPSE is an open source platform-independent project and has an architecture based on plugins. One of these plugins is the UML2Tools plugin, which supports the formal definition of UML profiles. Figure 10 shows how UML2Tools has been adapted by customizing the stereotype visualisation, thus enabling it to conform to the non-normative notational extension (Section 5.3), menus and palette tools.

The implementation of the application example (see Section 6) in the designed platform is also shown in Figure 10. The left-hand side of the figure represents the *Ticket Sale System* MRM diagram, modelled with the GRL profile for MRM, together with the palette tool for creating the modelling elements following the proposed notation. The centre of this same Figure shows the model-to-model transformations that join

the diagramming and interchange layers (Section 5). These transformations have been specified in QVT (Query/View/Transformation) language [59], and then codified through ATL (Atlas Transformation Language) [60]). In the centre of the Figure there is also a tab that shows the actual (*i.e.*, raw) implementation of MRM with EMF, in which we can observe the metaclasses used to represent the model itself (*Model*), its modelling elements (*Element*) and a particular element, which in this case corresponds to the link between the requirements and measurement subparts of MRM (*MeasuredThrough*). On the right-hand side of Figure 10, we can observe the textual model corresponding to the *Ticket Sale System* (see Section 6). For reasons of legibility, this textual MRM model has been divided into three parts: the requirements submodel is presented in the top window, while the requirements-measurement linkage is depicted in the middle window and the measurement submodel is shown in the bottom window. EMF directly supports this tree-like textual representation, which corresponds to the object diagram shown in Figure 9(a). Modelling elements and some useful properties such as containment relationships, data types, or generalisations, are are shown in this view. In order to complete the models specification, *Eclipse* also provides a properties view (in the lower part of the figure), which shows the properties of the selected classes.

It is important to note how the modelling architecture that implements *Eclipse* preserves the two-layer separation of our approach (see Figure 3) by providing two different kinds of metadata: on the one hand, the diagramming layer (based on UML profiles) is kept in XMI files, which store visualisation information (*e.g.*, nodes, arcs, 2D positions) together with the references to the rendered modelling elements. On the other hand, the actual model data (*i.e.*, the data obtained from the direct instantiation of the concepts defined in the raw MRM metamodel) is stored in XMI files that support EMF-based model persistence, which enables modellers to share MRM models among different types of model management tools (*e.g.*, diagramming tools, checking & validation tools, transformation engines, model repositories), through our interchange layer.

8 Conclusions and Further Work

The lack of consensus with regard to the use of late requirements metamodels in MDE proposals, together with the absence of mechanisms for specifying the requirements validation over other early development artefacts, is a reality that is hampering the alignment of MDE-based development projects with regard to stakeholders' needs. In order to alleviate this situation, we have proposed a 'measurable requirements metamodel' (MRM) which will assist MDE developers in the systematic elicitation of requirements, while simultaneously contributing towards obtaining the benefits of an early requirements evaluation. The support that MRM offers for the explicit linkage of goals, requirements and measures, helps to achieve both the backward and forward traceability of requirements. Moreover, in view of the fact that sound model rendering mechanisms are crucial for the success of a modelling framework, we have presented an associated UML profile, based on GRL and some well known diagramming principles, that simplify the task of modelling. The UML scaffolding provides MRM models with a degree of scalability (mainly based on the concept of package) that is missing in some well known GMLs, including GRL. This task is further simplified by an *Eclipse*-based tool that supports the MRM framework. Furthermore, the *Eclipse* environment allows MRM models to be integrated with MDE transformations that automate the measurement activities, thus speeding up the requirements validation process.

The definition of visual notations for modelling languages by profiling UML raises interesting research questions such as the usability evaluation of the software artefacts designed [50]. A significant challenge will therefore be the provision of a sound empirical evaluation that proves the suitability of the designed MRM diagrams for requirements elicitation and validation. In fact, despite its popularity, one of the main risks in the success of this evaluation is the aforementioned complexity of GMLs. Visual enhancements for MRM diagrams have been envisaged (such as the abstraction achieved by collapsing stakeholders' rationales), but the evaluation of such techniques requires further work.

References

1. Yamamoto, S., Kaiya, H., Cox, K., Bleistein, S.J.: Goal oriented requirements engineering: Trends and issues. IEICE Transactions **89-D** (2006) 2701–2711
2. Lapouchnian, A.: Goal-oriented requirements engineering: An overview of the current research. Technical report, University Of Toronto (2005)

3. van Lamsweerde, A.: Goal-oriented requirements engineering: From system objectives to UML models to precise software specifications. In: ICSE. (2003) 744–745
4. Dardenne, A., van Lamsweerde, A., Fickas, S.: Goal-directed requirements acquisition. Science of Computer Programming **20** (1993) 3–50
5. Mylopoulos, J., Yu, E., Nixon, B.A., Chung, L.: Non-Functional Requirements in Software Engineering. The Kluwer International Series in Software Engineering (2000)
6. Martínez, A., Pastor, O., Mylopoulos, J., Giorgini, P.: From early to late requirements: A goal-based approach. In: AOIS. (2006) 123–142
7. Fenton, N.E., Pfleeger, S.L.: Software Metrics: A Rigorous and Practical Approach. PWS Publishing Co., Boston, MA, USA (1998)
8. ISO: SPICE (Software Process Improvement and Capability dEtermination) (2004)
9. Chrissis, M.B., Konrad, M., Shrum, S.: CMMI: Guidelines for Process Integration and Product Improvement. Addison-Wesley (2006)
10. Larman, C.: Applying UML and Patterns. (2005)
11. Schmidt, D.: Guest editor's introduction: Model-driven engineering. IEEE Computer **39** (2006) 25–31
12. Selic, B.: MDA manifestations. The European Journal for the Informatics Professional IX (2008)
13. Kleppe, A., Warmer, J., Bast, W.: MDA Explained: The Model Driven Architecture: Practice and Promise. Addison-Wesley, Boston, MA, USA (2003)
14. Briand, L., Morasca, S., Basili, V.: Defining and validating measures for object-based high-level design. IEEE Trans. Softw. Eng. **25** (1999) 722–743
15. Object Management Group: Unified Modeling Language (UML), version 2.1.1. **http://www.omg.org/technology/documents/formal/uml.htm** (February 2007)
16. Lange, C., Chaudron, M., Muskens, J.: In practice: UML software architecture and design description. IEEE Software **23** (2006) 40–46
17. University of Toronto: Goal-Oriented Requirement Engineering Methodology Using GRL. **http://www.cs.toronto.edu/km/GRL** (Last visited October 2009)
18. Damian, D., Chisan, J.: An empirical study of the complex relationships between requirements engineering processes and other processes that lead to payoffs in productivity, quality, and risk management. IEEE Trans. SW Eng. **32** (2006) 433–453
19. Rashid, A., Moreira, A.M., Tekinerdogan, B.: Special issue on early aspects: aspect-oriented requirements engineering and architecture design. IEE Proceedings - Software **151** (2004) 153–156
20. Kotonya, G., Sommerville, I.: Requirements engineering with viewpoints. Software Engineering Journal **11** (1996) 5–18
21. Robertson, S., Robertson, J.: Mastering the requirement process. Addison-Wesley, Boston, MA, USA (1999)
22. Cockburn, A.: Writing Effective Use Cases. Addison-Wesley (2001)
23. Escalona, M.J., Aragón, G.: Ndt. a model-driven approach for web requirements. IEEE Trans. on Software Engineering **34** (2008) 377–390
24. Koch, N., Zhang, G., Escalona, M.J.: Model transformations from requirements to web system design. In: ICWE. (2006) 281–288
25. Bolchini, D., Paolini, P.: Goal-driven requirements analysis for hypermedia-intensive web applications. Requirement Engineering Journal **9** (2004) 85–103
26. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: Tropos: An agent-oriented software development methodology. Autonomous Agents and Multi-Agent Systems **8** (2004) 203–236
27. Potts, C., Takahashi, K., Antón, A.I.: Inquiry-based requirements analysis. IEEE Software **11** (1994) 21–32
28. International Telecommunications Union's Telecommunication Standardization Sector (ITU-T): User Requirements Notation (URN)/Z.151 metamodel. **http://jucmnav.softwareengineering.ca/twiki/bin/view/UCM/DraftZ151Metamodel** (September 2008)
29. Jureta, I., Mylopoulos, J., Faulkner, S.: Revisiting the core ontology and problem in requirements engineering. In: RE. (2008) 71–80
30. Kitchenham, B.: Guidelines for performing systematic literature reviews in software engineering. EBSE Technical Report EBSE-2007-01, Software Engineering Group, School of Computer Science and Mathematics, Keele University, UK and Department of Computer Science, University of Durham, Durham, UK (2007)
31. Molina, F., Pardillo, J., Cachero, C., Toval, A.: A systematic review of requirements metamodels. Technical report, University of Murcia. Available on http://www.lucentia.es/index.php/MRM (2008)
32. Herre, A.J.: Comet (component and model based development methodology). http://modelbased.net/comet/ (2006)
33. Fernández, J.L., Monzón, A.: A Metamodel and a Tool for Software Requirements Management. In: Reliable Software Technologies. Ada-Europe. Berlin (Germany). (2000)
34. Goknil, A., Kurtev, I., van den Berg, K.: A metamodeling approach for reasoning about requirements. In: ECMDA-FA. (2008) 310–325

35. Vicente-Chicote, C., Moros, B., Toval, A.: Remm-studio: an integrated model-driven environment for requirements specification, validation and formatting. Journal of Object Technology, Special Issue TOOLS EUROPE 2007 **6** (2007) 437–454
36. Vogel, R., Mantell, K.: MDA adoption for a sme: evolution, not revolution - phase ii. In: 2nd Workshop From code centric to model centric software engineering: Practices, Implications and ROI. In conjunction with ECMDA 2006. (2006)
37. Toval, A., Moros, B., Nicolas, J., Lasheras, J.: Eight key issues for an effective reuse. Int. Journal of Computer Systems Science **23** (2008)
38. I.S.O.: Iso/iec 9126-1. softw. eng.-product quality - part 1: Quality model. (2000)
39. Serrano, M.A., Calero, C., Sahraoui, H.A., Piattini, M.: Empirical studies to assess the understandability of data warehouse schemas using structural metrics. Software Quality Journal **16** (2008) 79–106
40. Blaine, J.D., Cleland-Huang, J.: Software quality requirements: How to balance competing priorities. IEEE Softw. **25** (2008) 22–24
41. Glinz, M.: A risk-based, value-oriented approach to quality requirements. IEEE Software **25** (2008) 34–41
42. Cachero, C., Melia, S., Genero, M., Poels, G., Calero, C.: Towards improving the navigability of web applications: a model-driven approach. European Journal of Information Systems **16** (2007) 420–447
43. García, F., Bertoa, M.F., Calero, C., Vallecillo, A., Ruiz, F., Piattini, M., Genero, M.: Towards a consistent terminology for software measurement. Information & Software Technology **48** (2006) 631–644
44. Briand, L., Morasca, S., Basili, V.: Property-based software engineering measurement. IEEE Trans. on Soft. Eng. **22** (1996) 68–86
45. Moody, D.: What makes a good diagram? improving the cognitive effectiveness of diagrams in is development. In: Advances in Information Systems Development, Springer (2006) 481–492
46. Object Management Group: Meta Object Facility (MOF) v2.0. **http://www.omg.org/mof/** (January 2006)
47. OMG: Object management group. (http://www.omg.org/)
48. Whitley, K.N.: Visual Programming Languages and the Empirical Evidence For and Against. J. Vis. Lang. Comput. **8** (1997) 109–142
49. Shneiderman, B., Card, S., Mackinlay, J.: Readings in Information Visualization: Using Vision to Think. Morgan Kaufmann (1999)
50. Green, T., Petre, M.: Usability Analysis of Visual Programming Environments: A Cognitive Dimensions Framework. Journal of Visual Languages and Computing **7** (1996) 131–174
51. Horkoff, J., Elahi, G., Abdulhadi, S., Yu, E.: Reflective analysis of the syntax and semantics of the i* framework. In: ER Workshops. (2008) 249–260
52. Amyot, D., Mussbacher, G.: Urn: Towards a new standard for the visual description of requirements. In: SAM. (2002) 21–37
53. Moody, D.: Comparative evaluation of large data model representation methods: The analyst's perspective. In: ER. (2002) 214–231
54. Mazón, J.N., Pardillo, J., Trujillo, J.: A Model-Driven Goal-Oriented Requirement Engineering Approach for Data Warehouses. In: ER Workshops. (2007) 255–264
55. Larkin, J.H., Simon, H.A.: Why a diagram is (sometimes) worth ten thousand words. Cognitive Science **11** (1987) 65–100
56. Warmer, J., Kleppe, A.: The Object Constraint Language: Getting Your Models Ready for MDA. Addison-Wesley, Boston, MA, USA (2003)
57. Gómez, J., Cachero, C., Pastor, O.: Conceptual modeling of device-independent web applications: Towards a web engineering approach. IEEE Multimedia **8** (2001) 26–39
58. Eclipse: Eclipse Modeling Framework Project (EMF). http://www.eclipse.org/modeling/emf/ (2008)
59. OMG: MOF QVT Standard Specification, http://www.omg.org/docs/ptc/05-11-01.pdf. (2005)
60. Team, A.: Atlas transformation language (atl). **http://www.eclipse.org/gmt/atl/** (Last visited October 2009)
61. Object Management Group: Object Constraint Language (OCL), version 2.0. **http://www.omg.org/technology/documents/formal/ocl.htm** (May 2006)

A List of Acronyms

EMF Eclipse Modelling Framework [58]
FR Functional Requirement
GDRE Goal-Driven Requirements Engineering
GML Goal Modelling Language
GORE Goal-Oriented Requirements Engineering
GRL Goal-oriented Requirements Language [17]

MD{A,E} Model-Driven Architecture [13]/Engineering [11]
MM Measurement Metamodel
MRM Measurable Requirements Metamodel (see Section 4)
MOF Meta-Object Facility [46]
NFR Non-Functional Requirement
OCL Object Constraint Language [61]
OMG Object Management Group [47]
QVT Query/View/Transformation language [59]
RE Requirements Engineering
RM Requirements Metamodel
SM{M,O} Software Measurement Metamodel/Ontology [43]
ULR Unified Late-Requirements metamodel (see Section 3)
UML Unified Modelling Language [15]
URN User Requirements Notation [52]

Artículo en *Journal of Universal Computer Science*

Journal of Universal Computer Science, vol. 15, no. 1 (2009), 100-200
submitted: s/x/09, accepted: s/x/09, appeared: 28/x/09 © J.UCS

ModelSec: A Generative Architecture for Model-Driven Security

O. Sánchez, F. Molina, J. García-Molina, A. Toval
University of Murcia (Spain)
osanchez@um.es, fmolina@um.es, jmolina@um.es, atoval@um.es

Abstract: Increasingly, the success of software systems depends largely on how their security requirements are satisfied. However, developers are challenged in implementing these requirements, mainly because of the gap between the specification and implementation, and the technical complexities of the current software infrastructures. Recently, Model-Driven Security has emerged as a new software development area aimed at overcoming these difficulties. This new paradigm takes advantage of the benefits of the model driven software development techniques for modeling and implementing security concerns. Following this trend, this paper proposes a model driven security approach named ModelSec that offers a generative architecture for managing security requirements, from the requirement elicitation to the implementation stage. This architecture automatically generates security software artifacts (e.g. security rules) by means of a model transformation chain composed of two-steps. Firstly, a security infrastructure dependent model is derived from three models, which express the security restrictions, the design decisions and the information needed on the target platform. Then, security software artifacts are produced from the previously generated model. A Domain-Specific Language for security requirements management has been built, which is based on a metamodel specifically designed for this purpose. An application example that illustrates the approach and the Eclipse tools implemented to support it are also shown.

Key Words: Requirements Engineering, Requirements Metamodelling, Model Driven Engineering, Model Driven Security

Categories: D.2.1

1 Introduction

Security is a crucial aspect in current software systems. Thus, security requirements must be adequately considered in all the phases of software development, from the consideration of requirements to the system implementation [Fernández-Medina et al., 2009, Haley et al., 2008]. To address the complex issue of developing secure systems which satisfy the desired security requirements, several model-based development approaches have recently appeared [Basin et al., 2006, Jurjens, 2003, Reznik et al., 2007, Lang and Schreiner, 2008]. Models have been traditionally used in Software Engineering as an abstraction mechanism for the analysis and design of software systems. For many years, their usefulness was mainly focused on documenting and considering the system to be built but, in this decade, they have become first-class software components, since source code can be generated from models specified by well-defined modeling languages.

Model-driven Engineering (MDE) [Selic, 2008] has emerged as a new area in software engineering, whose goal is the definition of theories, methods, techniques and tools for applying this model-based development paradigm. The Model Driven Architecture (MDA) initiative [OMG, 2003], promoted by the OMG, is the most popular MDE approach, although there are others such as Domain-Specific Development or Software Factories. Model driven approaches are based on three main principles: i) The creation of modeling languages (also named Domain Specific Languages, DSL) by applying metamodelling concepts, ii) The use of these languages to model different aspects of a software system, and iii) The automatic processing of the models built, by means of model transformations in order to generate software artifacts (e.g. source code) that will be part of the final application. On the other hand, most MDE tools are currently integrated in the Eclipse platform [Eclipse, 2008a] that provides implementations of the OMG specifications, for example, the Ecore metamodelling language. Besides, the definition of new domain specific languages is a more widely used practice than building UML profiles since the profiling mechanism allows for a limited extension form [Kelly and Tolvanen, 2008].

Two seminal works [Jurjens, 2003, Basin et al., 2006] laid the foundations for the application of the model-based development to the building of secure systems. Firstly, Jurjens proposes UMLSec as an UML extension aimed at expressing and evaluating specifications for vulnerabilities using formal semantics. Later, Basin et al. show how the MDE approach can be specialized to deal with security needs, namely Model-Driven Security (MDS). They illustrate the feasibility of the new MDS approach by using models to create software artifacts related to role-based access control infrastructure. A UML profile named SecureUML was built to support visual modeling of access control requirements. It is worth noting that UMLSec has evolved into an MDS approach [Jurjens et al., 2008].

In this paper we present an MDS approach, named ModelSec, which is based on current MDE practices and tools. ModelSec proposes a generative architecture for automatically generating security software artifacts (e.g. rules implementing security policies or security code for a database) from security models. This architecture is based on a model transformation chain composed of two steps. Firstly, a model, dependent on the security infrastructure, is derived from three models, expressing the security requirements, the design decisions made for implementing them, and the information needed on the target platform. Instead of UML profiles, a graphical DSL for expressing security requirements models has been defined, which conforms to a metamodel that includes the more common concepts related to the security aspects of a software system. In the second step, a model-to-code transformation generates software artifacts from the previously generated model.

ModelSec provides a significant advantage compared to the existing approaches: the proposed process enables developers to apply a systematic process. Modeling security requirements are separated from representing the design decisions for their implementation and from specifying the information related to the target platform needed for generating software. To accomplish this, a security model is created at the analysis, design and implementation stages of the development process. A novel feature of ModelSec is the possibility of modeling design choices, so that the developers have to specify in separate models *what* security requirements must be satisfied and *how* they are implemented. On another hand, ModelSec is a general MDS approach, and not centered on any specific aspect of security (e.g. access control). To achieve this, the DSL defined for expressing security requirement models allows the representation of seven different security aspects, and can also be easily extended in order to consider any new kinds of security requirements. Besides, the gap between the security requirements and the implementation code is reduced by introducing an intermediate model, which is the target platform specific model mentioned above.

The approach and the tools implemented on Eclipse to support it will be illustrated through an example of an application for medical information management, which shows how code for Oracle Label Security [ORACLE, 2008] and XACML [OASIS, 2008] policies can be generated from security models. It is important to note that we show how ModelSec enables us to deal with two different security aspects, but that other aspects could also be addressed in a similar way.

The remainder of the paper is organized as follows. Section 2 presents an overview of the proposed MDE. Section 3 shows our proposal for security requirement metamodelling. Sections 4 and 5 explain in detail the DSL implemented and how the architecture works as a whole. In Section 6, an example that further illustrates the use of our approach and its automatic support is shown. Section 7 analyses some related work. Finally, in Section 8, the main conclusions are drawn and further lines of research are outlined.

2 A model driven approach for security requirements

This section presents an overview of our understanding of how model-based development should be specialized in order to deal with security aspects. The proposed generative architecture in ModelSec is intended to support this vision. Figure 1 shows a schema of the designed process with the different models involved. As can be observed, the system security is represented by specific models which are separate from models concerning the rest of the requirements. At each stage of the development process a new security model is created, taking into account the model of the previous stage. These models are used to automatically generate a model dependent on the concrete platform chosen and, finally, code

of the final application can be produced. Next, the purpose of each model and the relationships among them is described.

During the requirements analysis stage, the definition of two separate models is proposed. Firstly, the developer creates a model for capturing the requirements of the system. Then, security requirements are represented in a more detailed way using a different model. Both models conform to the metamodel described in Section 3, but while the *requirements model* is focused on the non-specific requirements of the system, the *security requirements model* uses the security concepts of the metamodel (e.g. assets or threats) in order to appropriately represent the security requirements. This metamodel comes with a DSL aimed at creating security requirement models. Commonly, use cases and conceptual models are used to represent functional requirements. The requirement models proposed are not intended to replace the textual descriptions of the use cases, but are complementary. These models are needed to integrate requirements in an MDE process for which the use cases templates, expressed in natural language, are not suitable.

On the other hand, domain or conceptual models defining the domain vocabulary are often represented by class models (e.g. UML class models). Since conceptual models include all the assets, and therefore, the assets that need to be secured, a relationship among *security requirements models* and *domain models* is established: an asset of a security model maps and a concept in a domain model. This mapping will be used in the generative architecture to automatically generate the instances of the assets from classes in the domain model.

When developers consider the system design, they must make design decisions about how to implement every security requirement. These decisions (e.g., concerning security protocols and technologies) are specified in a *security design model*, which is created in two steps. First, a skeleton model is generated from a security requirement model using model transformation, where a security requirement is often expected to be mapped to the security decision that gives a technological solution for it. Then, developers must complete the model with information related to the design of the system security. *Security design models* do not include specific information about concrete protocols or technologies, but refer to abstract platforms (e.g. EJB). A *security implementation model* is another kind of platform dependent model aimed at representing this kind of information on the concrete platform used to implement the security requirements (e.g. an implementation of EJB as BEA WebLogic). It is worth noting that *security design models* and *security implementation models* are platform dependent models, whereas *security requirement models* should be independent of any platform. Therefore, in the same way as with the MDA approach, our proposal differentiates between platform independent models (PIM) and platform specific models (PSM). In Section 6, specific examples of these models will

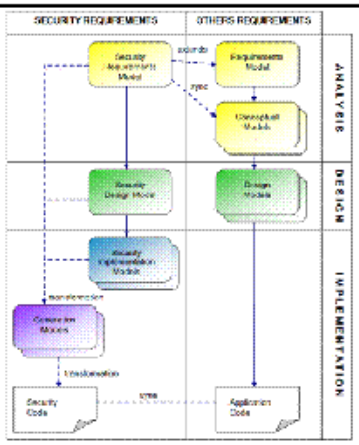

Figure 1: Overview of the models proposed by SecureModel

be shown through an example of an application.

The security models presented above are the input to a model-to-model transformation whose result is a target platform model that conforms to a metamodel representing a target technology. Security code of the final application is then generated from this intermediate model. There will be a target platform model, and therefore a security design model and a security implementation model, for each security aspect considered for generating software. There are some reasons for the existence of these intermediate models. Firstly, the semantic gap between the business requirements and the implementation code is reduced and a large transformation is replaced by two that are simpler. The existence of different transformations promotes reuse since the target platform metamodels and model-to-code transformations can be used in several contexts. Another benefit is that these models can be used to modify the code that is going to be generated without the need of editing it by hand. Because the *security design model* and the *design models* are not synchronized, in a few cases it cannot be expected that generated code related to security would be completely synchronized with the existing code.

In Section 5 the above security models in the context of the generative architecture of ModelSec will be explained.

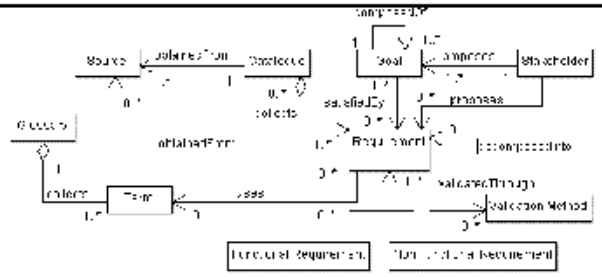

Figure 2: Proposed core requirements metamodel

3 A metamodel for security requirements

3.1 Requirements metamodelling

Most traditional approaches to Requirement Engineering (RE) use textual descriptions for the specification of requirements, which are organized in requirement documents that are rarely formally structured. Recently, the emergence of the MDE paradigm, especially the metamodelling technique, has introduced a new perspective for dealing with these requirements. Now metamodels are used to formally define the concepts and relationships involved in the analysis of requirements [Goknil et al., 2008].

Several approaches to metamodelling requirements have recently appeared [Goknil et al., 2008, Vicente et al., 2007, Berre, 2006]. However, a reference model has not yet been established and the existing approaches present a great disparity with regard to the number or semantics of the concerns addressed. Thus, a core requirement metamodel which contains the common concepts of the proposed metamodels and several new concepts and relationships considered relevant in our proposal (e.g. the concept of requirement reuse) has been defined. Moreover, these existing metamodels do not distinguish among different non-functional requirements, so an extension of the core metamodel, aimed at representing the real concepts of security requirements has also been defined. Figure 2 shows the core metamodel while Figures 3 and 4 in Subsection 3.2 show the extension proposed for including security concepts.

The key element in Figure 2 is that of `requirement`, which can be described by using a set of attributes such as an `identifier`, its `type` and its `textual description`. Requirements can be classified as either `functional` or `non-functional`. Each requirement is connected to the set of `goals` that it attempts to satisfy. Each goal can be composed of other goals, and captures a

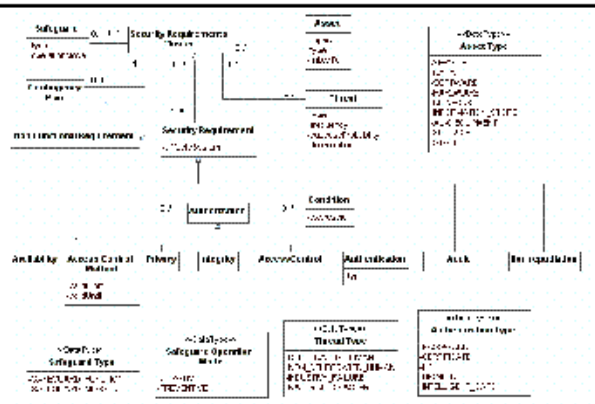

Figure 3: Extending the metamodel of Fig. 2: general security concepts

high-level objective from one or more **stakeholders** that propose it. Another concern is related to the reuse of requirements, which is tackled by including a catalogue concept and serves to gather a set of requirements extracted from one or more sources (*i.e.*, a law or a particular domain,) and that can be reused in all the projects to which these sources are applicable. A detailed explanation of the concepts on this core metamodel can be found in [Molina and Toval, 2009].

3.2 Extending the requirements metamodel with security concepts

The metamodel of Figure 2 has been extended with specific security concepts in order to define a DSL for security requirements. This extension has been divided into two modules in order to separate the access control mechanisms from the rest of the concepts (see Figures 3 and 4). The reason for this separation is that it is expected it will widen the range of control access mechanisms in the foreseeable future, so changes made to this part of the metamodel should not have side effects upon the rest of the metamodel. To facilitate the explanation, the concepts have been divided into three categories: basic security concepts, security requirements, and access control methods.

The basic security concepts are `Asset`, `Threat`, `Safeguard` and `Contingency Plan`. These terms conform to the standard ISO/IEC 15408 [ISO, 2005]. An `Asset` is a physical or logical object that has value in itself and deserves to have some guarantees with it. Assets can be of different types, for instance, documents, data tables, and so forth, and they have some importance for a business,

which is measured by an impact index. Assets refer to elements in the conceptual models specified by the `linkedTo` attribute, which can be implemented in two main ways. An option is to implement it as an explicit reference to a concrete model. Reference to a UML class diagram could be possible, given that an implementation of the UML metamodel exists in Ecore (i.e. the metamodel language on the platform Eclipse used to implement our approach). Another possibility could be to use identifier-based links, i.e. an identifier string that will match an identifier in a conceptual model. The first option allows us straightforward navigation through the conceptual model, while the second one allows us to decouple both models, although consistency between related elements in both models is not ensured. The latter option has been chosen so that any conceptual model can be used, not only concrete models such as UML class diagrams.

Assets can be damaged by a `Threat`, which has properties such as type, frequency (modeled as an annual rate), a probability of concrete success and degradation (that is, the level of damage caused in an asset if a threat achieves its goal). `Safeguards` serve as an impediment to a risk, in order to reduce it. As is shown in the type attribute, `Safeguard Functions` and `Safeguard Measures` are distinguished. The former are actions which reduce the risk whereas the latter are physical or logical devices or processes that reduce the risk. Two operational modes are distinguished for the safeguards: preventive if they act before a threat has taken place and curative if they act on damaged assets. For the sake of reducing a threat that can give rise to damage, a detailed `Contingency Plan` composed of a set of safeguards is recommended.

A standard classification for security requirements does not exist, so based on [Rodríguez et al., 2007], seven categories have been considered, which tackle seven categories of threats. These categories are: privacy, integrity, access control, authentication, availability, non-repudiation and auditing. Privacy consists of ensuring that information only can be read by those who are allowed. Integrity is the guarantee that the information stays complete and correct. Access Control is used to constrain the users that are authorized to access to an asset. These three kinds of requirements are related to `authentication` and can have a `condition`, defined as an expression. Authentication refers to the parties involved in a communication or interaction. Two kinds of authentication can be defined: authentication of the users of the service, also known as target authentication, and authentication of the data source, which is called source authentication. Availability consists of ensuring that authorized users have access to the information when needed. Non-repudiation is the ability to prove that parties have been involved in a communication or action, in such a way that parties cannot deny using the system. Finally, audit tries to log the use of services and data. All these sorts of requirements can affect either particular assets or the entire system.

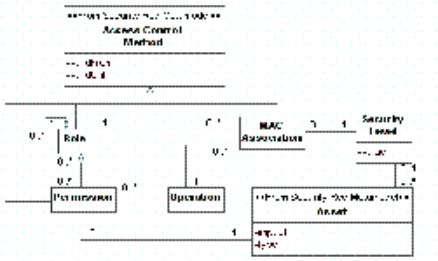

Figure 4: Extending the metamodel of Fig. 2: access control mechanisms

Frequently, a set of requirements exist which are related to the same asset, reduce the effects of the same attack and achieve the same security objective. This concept, extracted from [Mellado et al., 2007], is introduced in our metamodel as a `Security Requirements Cluster`. Regarding privacy, integrity and access control requirements, they are directly associated with an `Access Control Method` which has a period of validity. The different methods considered are `permissions` (Discretionary Access Control, DAC), `security levels` (Mandatory Access Control, MAC) or Roles (Hierarchical Role-Based Access Control, HRBAC) [Samarati and Capitani, 2000]. The concept of `MAC Association` has been introduced for associating a security level and an operation with a role. Note that security levels and roles are related, instead of security levels and users. Users may change from time to time so it is not desirable to model users at such a high level.

Bearing the above security concepts in mind, a security requirement metamodel has been obtained by extending the core metamodel shown in Figure 2 with the specific security concepts shown in Figures 3 and 4. These two figures only show the most important attributes of the concepts, and data types in Figure 3 only include some of the possible values. As can be seen, the extension point of the core metamodel to add new non functional requirements is the `NonFunctional Requirement` metaclass. `Security Requirement` is a specialization of `Non-Functional Requirement` intended to be root for the metaclasses representing security concepts.

4 A DSL for security requirements

In this section the DSL created to express the models which represent the security requirements in a simple and intuitive way is explained. These models are named *security requirement models* and they define the security needs to be satisfied,

in such a manner that platform details are not described. While most existing approaches provides UML profiles (e.g. UMLsec and SecureUML), our approach provides a DSL tailored to this domain and called SecML (Security Modeling Language).

Nowadays, the implementation of a DSL is considered more appropriate than defining an UML profile, because their design and implementation is not restricted by the limitations of the profiling mechanism [Kelly and Tolvanen, 2008, Voelter, 2008]. When metamodelling techniques are applied, a DSL consists of three elements: an abstract syntax metamodel, concrete syntax, and semantics. The abstract syntax defines the concepts that take part in the DSL and the relationships between them, and also includes the rules which constrain how the models can be created. The concrete syntax defines a notation for the abstract syntax, and semantics is normally provided by means of model transformations which transform a model expressed in the DSL into models expressed in languages with well-defined semantics [Kelly and Tolvanen, 2008].

In our case, the abstract syntax is given by the metamodels explained in Section 3. The graphical representation of the concepts (i.e. the concrete syntax) included in the DSL is as follows: (see Figure 8(a)). Basic security concepts, security requirements and access control methods in the metamodel are depicted by a rectangle with an icon in the upper-left corner. Each concept has a different icon. For example, the clusters of security requirements have a padlock, security requirements have a star of a different colour, depending on their type, and assets are depicted with three small balls. Different from other elements, clusters have compartments in which the security requirements are included. Some of the relationships between concepts, such as the relationships between cluster and threat or asset, have been explicitly represented by means of arrows. Finally, model-to-model transformations from *security requirements models* to software artifacts give semantics to our DSL.

The choice of a graphical DSL instead of a textual one has been determined by the fact that non-expert users could use it. Graphical DSLs are more readable and meaningful for representing relationships between concepts, such as relationships between requirements and assets or, in our case, threats. In addition, they are often easier to learn than textual DSL.

SecML plays a key role in our generative architecture. SecML models can be created by users who only have business knowledge. The analyst can express the security restrictions by using well-known domain concepts, while the modeling of design decisions and implementation details are left to the experienced developers. A SecML specification has been split into two views: a view for modeling textual requirements and a view specifically designed to model security requirements with SecML. The latter view is related to the former since it details some of the requirements specified in the textual view. SecML has been imple-

mented on Eclipse using the Graphical Modeling Framework (GMF), a powerful, practical and widespread framework for giving a graphical concrete syntax to an abstract syntax expressed with Ecore metamodels. As we indicated, Eclipse is a well known platform that provides, through several projects, the most widely used tools and frameworks for MDE. Moreover, GMF has several interesting features: it is open source like Eclipse, which means that anybody can customize it; a DSL created with GMF takes advantage of the tools and features of the Eclipse IDE; code generation is supported by different template languages; and, tool interoperability is obtained by using XMI, the model serialization format of Eclipse.

5 From security models to security software artefacts

This section explains how the ModelSec generative approach works. Previously described security requirement models are completely platform independent and, thus, other models are needed to represent platform specific security data. Therefore, previous to presenting an overview of the approach, these models will be described.

5.1 Making design decisions: the Security Design Model

Whereas *security requirements models* specify *what* security restrictions the final system must satisfy, *security design models* specify *how* these restrictions will be satisfied. These models conform to the security design metamodel shown in Figure 5, which represents the design decisions related to security requirements specified in SecML. The metamodel is simple, and contains a hierarchy of types of design decisions along with a set of data types. The hierarchy has a metaclass for each security aspect considered by the requirement specification (e.g. login, authorization, and access control policy). The description of the attributes of these metaclasses shows an idea of the type of information provided by means of design security models.

Each requirement in a security requirement model is mapped to an instance of `SecurityDesignDecision`. Currently, authentication requirements are mapped to `AuthenticationDecision`, non-repudiation and audit requirements are mapped to `LoggingDecision`, and authorization requirements are mapped to `AccessControlDecision`. Other security aspects available are not yet considered due to complexity in implementation in software. They will be considered in further work.

The `SecurityDesignDecision` metaclass has four attributes, representing properties common to any design decision. For each design decision, the list of requirements related to it must be specified. If a requirement implies the use

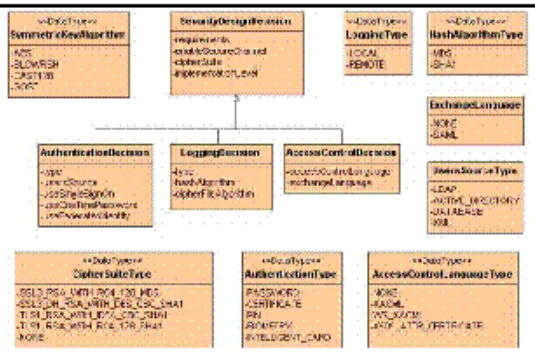

Figure 5: Security Design Metamodel

of a network, the `enableSecureChannel` attribute indicates whether the communications must be secured, and `cipherSuite` declares the set of protocols and security algorithms to use. This suite includes the cryptographic protocol, the public-key algorithm, the symmetric-key algorithm and the hash algorithm. The `implementationLevel` attribute lets the developer declare whether the requirement will be implemented by either a pre-defined or a custom solution. Depending on the solution, different artifacts could be generated. For example, authentication could be accomplished by a module server, and a skeleton of the XML server configuration file that includes authentication could be automatically generated. On the other hand, authentication could not rely on a third-party solution, and Java Authentication and Authorization Service (JAAS) [SUN, 2008] could be automatically generated.

Each metaclass inherited from `SecurityDesignDecision` adds its own attributes. An `AuthenticationDecision` includes a type of authentication that must be consistent with the security requirements model, if specified. The source of the users (i.e. `usersSource` attribute) can also be indicated. There are some other options available for authentication, such as `useSingleSignOn`, `useOneTimePassword`, `useFederatedIdentity` that allow us to customize any features of the authentication needed.

A `LoggingDecision` has a type of log mechanism, a `cipherFileAlgorithm` property that indicates whether the log file must be ciphered and which is the symmetric-key algorithm involved, and a `hashAlgorithm` property that specifies the algorithm used to perform a hash of the file, if desired.

With regard to the `AccessControlDecision` metaclass, the attribute `accessControlLanguage` is used to specify that we are interested in writing

control policies explicitly with a suitable language, and exchangeLanguage indicates that we would like to use an appropriate language for considering the access control petitions and decisions.

As can be observed, this metamodel is expected to change in the future, given that new protocols or technologies are constantly evolving, and new solutions (i.e. design decisions) will be considered. For this reason, this metamodel includes only the most common attributes we have detected at the present time.

5.2 Considering the platform: the Security Implementation Models

When design decisions have been taken into account, detailed information about them must be provided in order to accomplish a proper generation of software artifacts. This low-level information is dependent on the target platforms for which these artifacts are generated. This justifies why our approach distinguishes between security design models and security implementation models and avoids mixing design decisions and platform specific data. Since there are too many specific platform details at this level, metamodels are not shown, but some examples of them can be found in Section 6.

One or more security implementation models are usually created for each design decision. No *implementation model* will be defined if we are not interested in generating some kind of predefined artifact or there is not a current transformation available for the chosen technologies. For example, a policy model should be created if we defined an access control decision enabling the accessControlLanguage option. A skeleton of the implementation models can be obtained from the design models through model-to-model transformations, in the same way that a design model can be generated from the requirements model. As indicated in Section 2, *security design models* and *security implementation models* are platform specific models (PSM). Notice that the former is dependent on a platform at a more abstract level that the latter.

As a rule, in addition to the models derived from every design decision, an implementation parameter model is required. The goal of this model is to provide information about the whole target system, such as the type of application, the concrete target platform, the technologies implementing the different layers, etc. Moreover, this model allows us to decouple the declaration of application users from the requirement models. Since users are frequently expected to change throughout the life of the new system, it is desirable to declare them as late as possible so they will not affect the higher level models.

5.3 Generative architecture overview

Figure 6 shows an overview of the presented generative architecture, which has been implemented using Eclipse. As it was previously stated, once a security

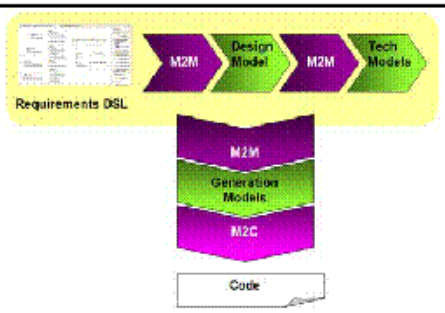

Figure 6: Overview of the generative architecture

requirement model is available, a *security design model* must be created, which depends on the previous model. In the same way, *security implementation models* must be defined based on the *security design model*. Making use of the generative approach, model-to-model transformations (M2M) have been defined for generating skeletons of these dependences. Thus, a *design model* skeleton is derived from a *security requirement model* that is created by means of the security requirements DSL. Similarly, *security implementation model* skeletons can be derived from the *security design model* by means of M2M transformations. This chain of model transformations can be seen at the top of Figure 6. Notice that design and implementation models are created by developers by adding information to the generated skeletons.

Another chain of model transformations can be observed in Figure 6, which is depicted vertically. This chain aims to automatically generate software artifacts from the previously created security models. A security requirement model, together with the security design and security implementation models are the input of a M2M transformation that generates several target platform models. RubyTL [Cuadrado et al., 2006] is the transformation language selected to implement these transformations, since it is compatible with Ecore, which allows us to interoperate with all of the tools implemented. Finally, model-to-code (M2C) transformations are used for generating software artifacts related to the system security. In this case, the MOFScript [Eclipse, 2008b] template language has been chosen due to it being a simple but powerful language based on the use of rules.

Note that the chain of model transformations of the generative architecture works due to the fact that all the tools used are compatible, in the sense that they serialize Ecore models in XMI format. Next section illustrates how this

architecture has been used for generating security code for Oracle and XACML policies.

6 Application example

To illustrate how the ModelSec approach works, the example adapted from [Fernández-Medina and Piattini, 2005] of a web application for the management of medical patients will be used. This example, which includes the design of a secure database, serves us to show how a role-based control access policy and database security code can be automatically generated. An in depth explanation of those models which are not related to security has been omitted since they are not needed to understand the example but, when necessary, the dependences among models will be shown.

As indicated in Section 2, the process starts with the definition of the requirements of the system using the SecML. Figure 7 shows the view of the tool provided for managing requirements, which is used by the analyst for editing the textual description of the requirements. Instead of directly creating a model, the analyst could use this view to introduce each requirement (description and type), and then the corresponding model is generated and populated with an instance for each requirement. Then, the analyst uses the graphical view of the tool for the input of the values of the properties of each requirement. In Figure 7 an extract of a catalogue of requirements elicited by a hospital manager related to the data of the hospital patients is shown. The upper part of the catalogue includes a pair of functional requirements (labeled as REQ_F1 and REQ_F2) whereas the lower part involves non-functional requirements (labeled from REQ_S1 to REQ_S7). For reasons of simplicity, only some non-functional requirements have been included. It is important to note that SecML could be used for modeling both functional and non-functional requirements.

An excerpt of the *security requirements model* for this catalogue appears in Figure 8(a), which shows a screenshot of the security view of the tool. The window panel consists of two parts: the editing area and the element palette. A model with different types of elements can be observed in Figure 8(a), while the properties that describe the most relevant types of elements are presented in Figure 8(b). In Figure 8 several elements are visualized: four clusters containing a total of seven security requirements, two assets, a hierarchy of roles, three threats, three access control policies, and a contingency plan. *Patient* and *MedicalHistory* are the main assets that need to be secured. The former represents the personal information while the latter is a record of the illnesses and medical treatment of a patient. These two assets reference classes in the conceptual model, which are specified by the `linkedTo` attribute (an explanation of how this attribute can be implemented was indicated in Section 2).

Figure 7: An example of security requirements catalogue

The CL1 cluster includes security requirements that restrict access to the system to the medical staff. In CL1, the requirement REQ_S1 restricts access to previously registered users who must provide Login/Password credentials, and REQ1_S2 states a constraint on access, since only those users with a staff role who attempt to access between 9 a.m. and 5 p.m. are allowed (this is specified in the condition attribute of REQ_S2 in Figure 8(b)). Date, time and IP addresses can be restricted by using the simple expression language defined. These requirements lead the analyst to create a role hierarchy whose root is the *Staff* role, and with two sub-roles named *Doctor* and *Nurse*, which will be used afterwards.

The CL2 cluster is intended to prevent unauthorized staff accessing patient information, and the CL3 cluster has a similar purpose, but related to *MedicalHistory*. The requirements contained in the cluster CL4 attempt to prevent unauthorized modification of the *patient* data.

With regard to the control access policy, the analyst has decided to use MAC for taking control over the users who attempt to query or modify patient data. Thus, *public*, *confidential* and *sensitive* elements have been added to the model. A *confidential* level has been assigned to the asset *Patient*, and *sensitive* to *MedicalHistory*. Requirements of privacy and integrity must specify a security level by considering the labels established for patient data and histories. It is important to note that although security levels are used, we need to specify which group of users will be granted this level, so roles are used for this purpose. For instance, REQ_S3 assigns the confidential read level on the patient data to the role *Doctor*, which means that the doctors can query the information related to the patients because its level is the same as the *Patient* asset.

In the example we can see that the clusters refer to three threats and one contingency plan. These two kinds of elements allow the specifying of information

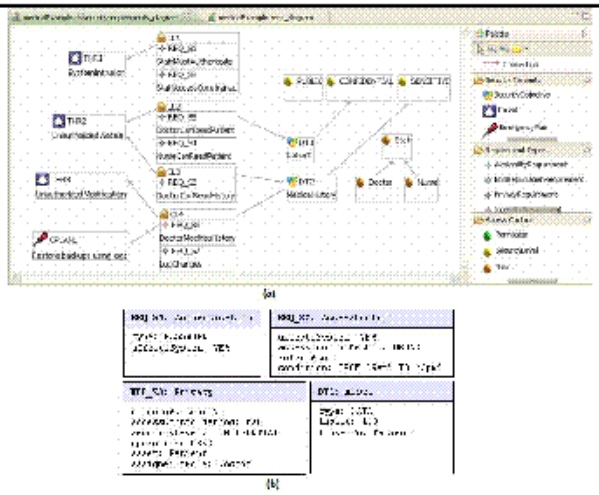

Figure 8: (a) Excerpt of a security requirements model (b) Attribute values

that can be used to generate system documentation, for instance, weaknesses and possible solutions. Although threats and contingency plan elements have been depicted in the requirement model, they will not be considered in the rest of the example because of lack of space.

Once the security requirement model is completed, a *security design model* must be created in order to express the design decisions. As explained previously, due to the existing mapping among elements of both models, a design model skeleton can be automatically generated from the requirement model, and then the developer specifies the information of the model elements. Following with our example, Figure 9(a) shows the four elements of this model: one `AuthenticationDecision`, two `AccessControlDecision` and, finally, one `LogginDecision`. Only those element properties that do not have the default value (i.e., are different from NO or NULL) have been shown. No secure communications have been included since they are not needed in this example.

The REQ_S1 requirement of type authentication is mapped to an instance of `AuthenticationDecision`. In REQ_S1, *password* is specified as the value of the attribute type, which means a login/password authentication. Therefore the value of the type attribute of the `AuthenticationDecision` element also has this value. The `implementationLevel` attribute has been established to *custom*,

because the authentication will not be automatically performed for any module, but we will implement this control in our application. We also specify that the users are registered in a LDAP server, but so long as we are going to implement authentication manually, this annotation does not have further implications.

A *custom* implementation level has also been specified for the element `AccessControlDecision`, which is linked to the REQ_S2 requirement of type `AccessControl`. This design decision is motivated by the fact that we intend to implement the `Policy Enforcement Point`, the `Policy Decision Point` and the whole authorization process by ourselves. As the access control policies need to be automatically generated, XACML has been assigned to the `accessControlLanguage` attribute in order to indicate the language for generating the policies. SAML assertions could also be generated, but this possibility has not been considered, in order to simplify the example.

A second `AccessControlDecision` example refers to the requirements from REQ_S3 to REQ_S6, and are those that prevent unauthorized reading or modification of any of the personal data or medical history of the patients (i.e. authorization requirements). In this case, the implementation level of the authorizations is expressed as database, which means that the access control method will be implemented in the database management system and it will be responsible for checking the access rights to the tables related to the assets.

Finally, a `LoggingDecision` element has been included for the REQ_S7 requirement of type auditing. The type attribute takes the value *local*, which indicates that we will use a local log without any kind of protection and, its implementation is the choice of the developer.

At this point, the *security design model* specifies how the security requirements will be implemented, but the information provided only indicates, at an abstract level, which technological solutions will be used, but does not includes details about target platforms. Thus, a *security implementation model* is used to express the implementation details that refer to a concrete target platform which are needed for generating software artifacts. In this example, the *security design model* created is detailed in three security implementation models which represent low-level data of the access control policy (Figure 9(b)), the security database (Figure 9(c)) and the target platform (Figure 9(d)). Note that we have a general access control policy model, which could be used with different access control languages, while we have a specific database model for Oracle. The reason for this disparity is that we cannot have a general database model, because each database system has its own security solutions, so specific models are needed.

The number and nature of the security implementation models depends on the technologies chosen, for which software artifacts are generated. In our case, a *policy model* has been defined to generate access control policies, and an *Oracle*

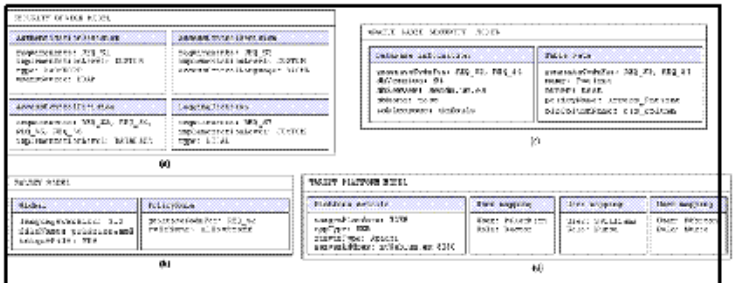

Figure 9: (a) Security design model, (b) Policy model, (c) Oracle model, (d) Target platform model

model to generate database code. A definition of an implementation model is always required for any generation in order to provide some details about the target platform, such as the application type, and a list of the users initially registered in the system. Regarding the *policy model*, a few global parameters must be specified, such as the target file and some data for each access control requirement. To illustrate this, a permission rule for REQ_S2 will be generated.

Although pair user-roles are defined in this model, it does not necessarily imply that roles will be defined in the database. In the example, we define roles because we need to assign security levels to users and we do not want to define users in the security requirements model. However, no role definition code will be generated. So, roles serve as a mechanism to decouple users and access control methods.

All models created are input to a M2M transformation specified in RubyTL that automatically generates target platform models. A model will be generated for each security implementation model. In this example, a model for access control policies and a model for database code will be generated. As indicated earlier, these models conform to the metamodels representing the target platform; for this example a XACML metamodel and an Oracle Label Security metamodel were defined. Finally, security software artifacts are automatically generated from target platform models by means of M2C transformations specified in MOFScript. In this case, the software artifacts created have been a XACML policy file and an Oracle PL/SQL script. It is remarkable that the target platform models as well as the code are automatically generated, and no human intervention is required.

Figure 10 shows an excerpt of the XACML policy file generated. This is a snippet of code that includes a rule that authorizes users classified as *Staff* to access the resources between 9 a.m. and 5 p.m. The description of the rule is the

Figure 10: Excerpt of the XACML policy file automatically generated

text of the security requirement itself. The subject of the rule (i.e. the role *Staff*) and the condition that restricts the hours of access are both extracted from the properties stated in the previous models. Figure 11 shows an excerpt of PL/SQL Oracle Label Security code. As we specified in the Oracle model (Figure 9(c)), a policy for controlling the reading of the *Patient* table has been generated. The different security levels are created as well as the labels connected to them. After this, the labels are assigned to the different users defined in the implementation target model, according to the reading and writing restrictions declared in the security requirement model. Finally the policy is applied to the table.

It is worth to remarking that the creation of security artifacts involves a great deal of text even though only simple requirements have been considered.

7 Related work

The approach presented can be related to works both on requirement metamodeling in the field of RE and MDS approaches. As mentioned in Section 3, different requirement metamodels have been proposed [Goknil et al., 2008, Berre, 2006, Vicente et al., 2007, Bolchini and Paolini, 2004]. These metamodels are focused on the basic concepts of the RE and do not include concepts related to concrete non-functional requirements such as, for example, security. We have defined a

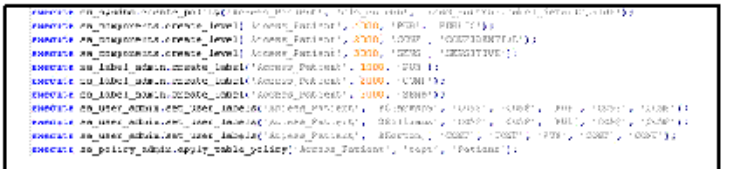

Figure 11: Excerpt of the Oracle automatically generated

requirement metamodel which combines a set of core requirement concepts with a set of security concepts. The core concepts have been obtained by identifying the concepts common to the existing metamodels. Moreover, most of the existing approaches for metamodelling requirements have not considered the definition of a concrete syntax (notation) or the automatic generation of software artifacts from requirements.

As indicated above, two of the most relevant approaches in security modeling are UMLSec [Jurjens, 2003] and SecureUML [Basin et al., 2006]. Both of them provide UML extensions to specify security requirements. Whereas UMLSec was conceived to verify formal security requirement specifications in the system design, SecureUML was used to illustrate the MDS approach. Although SecureUML is specific to role-based access control infrastructures, it shows how an MDE strategy could be applied to generating code for any aspect of security, and since then a number of MDS approaches have been proposed. For instance, [Reznik et al., 2007] presents an MDS solution to developing secure applications on a middleware platform which integrates an implementation of the Corba Component Model with the OpenPMF security framework. In this approach, another UML profile is created in order to model access control policies. [Lang and Schreiner, 2008] illustrates how the OpenPMF architecture can be used to translate a security-related high-level regulatory requirement into enforceable authorization rules. In this case, the example considered is similar to that presented in this paper: a healthcare security requirement is implemented using XACML.

Our proposal has several characteristics that differentiate it from existing MDS approaches. The generative architecture has been organized in two chains of model transformations, as shown in Figure 6. This organization promotes a systematic process for applying MDS, which separates specifications of security requirements and design decisions of how they are implemented. In contrast to other approaches, models are used to represent design decisions and implementation details about the target platforms. Model reuse is facilitated by this separation. In order to reduce the semantic gap among security requirements and generated software, ModelSec proposes an intermediate model which con-

forms to a target platform metamodel. This intermediate step promotes reuse of transformations. Instead of building UML profiles, as with the other approaches, a DSL for expressing security requirements has been defined, which has been implemented by applying metamodelling techniques. Moreover, ModelSec is a generic approach, while most MDS approaches have been focused on the access control policies.

There exist other approaches in the scope of security requirements management that are not aligned to MDS and focus mainly on the elicitation of security requirements more than in the generation of software artefacts from them. That is the case of proposals as Secure Tropos [Bresciani et al., 2007], [van Lamsweerde, 2004], that introduces the concept of antigoal, [Yu et al., 2007] that uses ontologies or [Haley et al., 2008], which presents a framework composed by a set of activities to deal with security requirements.

8 Conclusions and further work

Creating security software artifacts requires handling a large amount of text in formats such as source code or XML text. In a similar way to other MDS approaches, ModelSec offers a high level of automation of the generation of code aimed at dealing with the security requirements of the system (e.g. security policies), and which avoid a tedious, time consuming, costly and error-prone manual process. However, in a different manner to the existing MDS approaches, ModelSec proposes a generative architecture based on a chain of model transformations involving several security models at different levels of abstraction. Throughout this paper how this architecture promotes the reuse and provides a more systematic process than the existing approaches has been shown. Another significant contribution is that the DSL for security requirements is built on our own requirement metamodel.

As for further work, ModelSec will be extended by defining new target platform metamodels that serve as a basis for the generation of other access control and authorization policies. Moreover, ModelSec is being used for the generation of code for real software projects, and it is expected that there will be interesting feedback for the refinement and extension of this proposal.

9 Acknowledgements

This work has been partially supported by the projects DEDALO (TIN2006-15175-C05-03) and PANGEA (TIN2009-13718-C02-02) from the Spanish Ministry of Science and Technology, MELISA-GREIS (PAC08-0142-3315), 08797/PI/08 from the Fundación Séneca and 129/2009 from the Regional Council of Murcia. Fernando Molina is partially funded by the Fundación Séneca (Murcia).

References

[Basin et al., 2006] Basin, D., Doser, J., and Lodderstedt, T. (2006). Model driven security: From uml models to access control infrastructures. *ACM Trans. Softw. Eng. Methodol.*, 15(1):39–91.

[Berre, 2006] Berre, A. J. (2006). Comet (component and model based development methodology). http://modelbased.net/comet/.

[Bolchini and Paolini, 2004] Bolchini, D. and Paolini, P. (2004). Goal-driven requirements analysis for hypermedia-intensive web applications. *Requirement Engineering Journal*, 9(2):85–103.

[Bresciani et al., 2007] Bresciani, P., Mouratidis, H., and Zanone, N. (2007). Modelling security and trust with secure tropos. *Integrating Security and Software Engineering: Advances and Future Visions*, pages 160–189.

[Cuadrado et al., 2006] Cuadrado, J., Molina, J., and Tortosa, M. (2006). Rubytl: A practical, extensible transformation language. pages 158–172.

[Eclipse, 2008a] Eclipse (2008a). Eclipse Modeling Framework Project (EMF). http://www.eclipse.org/modeling/emf/.

[Eclipse, 2008b] Eclipse (2008b). Generative Modeling Technologies (GMT): MOFScript. http://www.eclipse.org/gmt/.

[Fernández-Medina et al., 2009] Fernández-Medina, E., Jurjens, J., Trujillo, J., and Jajodia, S. (2009). Editorial: Model-driven development for secure information systems. *Inf. Softw. Technol.*, 51(5):809–814.

[Fernández-Medina and Piattini, 2005] Fernández-Medina, E. and Piattini, M. (2005). Designing secure databases. *Information & Software Technology*, 47(7):463–477.

[Goknil et al., 2008] Goknil, A., Kurtev, I., and van den Berg, K. (2008). A metamodeling approach for reasoning about requirements. In *ECMDA-FA*, pages 310–325.

[Haley et al., 2008] Haley, C., Laney, R., Moffett, J., and Nuseibeh, B. (2008). Security requirements engineering: A framework for representation and analysis. *IEEE Trans. Software Eng.*, 34(1):133–153.

[ISO, 2005] ISO (2005). Iso/iec 15408 (common criteria v3.0): Information technology security techniques-evaluation criteria for it security.

[Jurjens, 2003] Jurjens, J. (2003). *Secure Systems Development with UML*. Springer Verlag.

[Jurjens et al., 2008] Jurjens, J., Schreck, J., and Bartmann, P. (2008). Model-based security analysis for mobile communications. In *ICSE*, pages 683–692.

[Kelly and Tolvanen, 2008] Kelly, S. and Tolvanen, J. (2008). *Domain-Specific Modeling: Enabling Full Code Generation*. Wiley-IEEE Computer Society Press.

[Lang and Schreiner, 2008] Lang, U. and Schreiner, R. (2008). Managing business compliance using model-driven security management. In *Securing Electronic Bussness Processes*, pages 1–11.

[Mellado et al., 2007] Mellado, D., Fernández-Medina, E., and Piattini, M. (2007). A common criteria based security requirements engineering process for the development of secure information systems. *Comput. Stand. Interfaces*, 29(2):244–253.

[Molina and Toval, 2009] Molina, F. and Toval, A. (2009). Integrating usability requirements that can be evaluated in design time into model driven engineering of web information systems. *Journal of Advances in Engineering Software*, 40:1306–1317.

[OASIS, 2008] OASIS (2008). "extensible access control markup language". "http://www.oasis-open.org/".

[OMG, 2003] OMG (2003). *MDA Guide v1.0.1*. http://www.omg.org/mda.

[ORACLE, 2008] ORACLE (2008). Oracle label security. http://www.oracle.com/technology/deploy/security/database-security/label-security/index.html.

[Reznik et al., 2007] Reznik, J., Ritter, T., Schreiner, R., and Lang, U. (2007). Model driven development of security aspects. *ENCTS*, 163(2):65–79.

[Rodríguez et al., 2007] Rodríguez, A., Fernández-Medina, E., and Piattini, M. (2007). A bpmn extension for the modeling of security requirements in business processes. *IEICE Transactions*, 90-D(4):745 752.

[Samarati and Capitani, 2000] Samarati, P. and Capitani, S. D. (2000). Access control: Policies, models, and mechanisms. In *FOSAD*, pages 137 196.

[Selic, 2008] Selic, B. (2008). Mda manifestations. *The European Journal for the Informatics Professional*, IX(2).

[SUN, 2008] SUN (2008). Java authentication and authorization service. http://java.sun.com/javase/technologies/security/.

[van Lamsweerde, 2004] van Lamsweerde, A. (2004). Elaborating security requirements by construction of intentional anti-models. In *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, pages 148 157, Washington, DC, USA. IEEE Computer Society.

[Vicente et al., 2007] Vicente, C., Moros, B., and Toval, A. (2007). Remm-studio: an integrated model-driven environment for requirements specification, validation and formatting. *Journal of Object Technology, Special Issue TOOLS EUROPE 2007*, 6(9):437 454.

[Voelter, 2008] Voelter, M. (2008). Md* best practices.

[Yu et al., 2007] Yu, E., Liu, L., and Mylopoulos, J. (2007). A social ontology for integrating security and software engineering. *Integrating Security and Software Engineering: Advances and Future Visions*, pages 70 109.

Artículo en *Information and Software Technology*

Information and Software Technology 51 (2009) 1631–1645

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

A systematic review of UML model consistency management ☆

Francisco J. Lucas *, Fernando Molina [1], Ambrosio Toval

Software Engineering Research Group, Department of Informatics and Systems, University of Murcia, Spain

ARTICLE INFO

Article history:
Available online 3 May 2009

Keywords:
UML
Model consistency
Systematic literature review

ABSTRACT

Information System (IS) development has been beset by consistency problems since its infancy. These problems are greater still in UML software development, and are principally caused by the existence of multiple views (models) for the same system, and may involve potentially contradictory system specifications. Since a considerable amount of work takes place within the scope of model consistency management, this paper presents a systematic literature review (SLR) which was carried out to discover the various current model consistency conceptions, proposals, problems and solutions provided. To do this, a total of 907 papers related to UML model consistency published in literature and extracted from the most relevant scientific sources (IEEE Computer Society, ACM Digital Library, Google Scholar, ScienceDirect, and the SCOPUS Database) were considered, of which 42 papers were eventually analyzed. This systematic literature review resulted in the identification of the current state-of-the-art with regard to UML model consistency management research along with open issues, trends and future research within this scope. A formal approach for the handling of inconsistency problems which fulfils the identified limitations is also briefly presented.

1. Introduction

Consistency problems have existed in Information System (IS) development since its beginning and are usually linked to the existence of multiple models or views which participate in the development process.

Problems related to the maintenance and management of consistency were identified in the design of the first databases. These problems were, on some occasions, due to the duplication of information as a result of design errors and, on others, caused by the need to denormalize this design in order to obtain a more efficient database execution.

When focusing on software development using UML (Unified Modelling Language) [1], problems related to consistency between models also appear. This is mainly due to the existence of multiple views (models) for the same system, which may potentially contain contradictory specifications. These inconsistencies among different models or views of a system may be a source of numerous errors in the software developed [2] and may, moreover, consequently complicate its management [3]. A proof of the importance of UML model consistency in software development is the existence of both a considerable amount of work within this scope (see Section 3.6), and specific workshops and conference sessions attempting to deal with this kind of problems.

Furthermore, in recent years, the profound impact of the Model Driven Engineering (MDE) [4] and particularly of the Model Driven Architecture (MDA) proposal [5], in which the use of models guides the development of a system, has caused models to gain more importance. According to [6], inconsistency problems have become critical within the scope of MDE and could make the use of models as a source of automatic code generation impossible.

Attempts to solve these problems have led to the appearance of a great deal of work in model consistency management. This paper presents a systematic literature review (SLR), which was carried out in order to discover the various current consistency conceptions, proposals, problems and solutions provided. A systematic literature review provides a means of identifying, evaluating, and interpreting the literature relevant to a particular research question or topic area [7]. There are numerous reasons for carrying out systematic literature reviews such as:

- To summarize the existing evidence concerning a treatment or technology, e.g. to summarize the empirical evidence of the benefits and limitations of a specific agile method.
- To identify any gaps in current research in order to suggest areas for further investigation.
- To provide a framework/background in order to appropriately position new research activities.

☆ Partially financed by the Spanish Ministry of Science and Technology, project DESARCO (TIN2006-15175-C05-03).

* Corresponding author.
E-mail addresses: fjlucas@um.es (F.J. Lucas), fmolina@um.es (F. Molina), atoval@um.es (A. Toval).

[1] Supported by the Fundación Séneca (Región de Murcia).

0950-5849/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2009.04.009

This SLR has been carried out by considering a total of 907 papers related to UML published in literature and extracted from the most relevant scientific sources (IEEE Computer Society, ACM Digital Library, Google Scholar, ScienceDirect, and the SCOPUS Database). The selected papers have been classified according to a number of features within the scope of model consistency management. In the context of this report, this systematic literature review attempts to answer the following question: What inconsistency problems have been tackled by the current approaches for model consistency management and how extensible are they in guaranteeing its usability and maintainability, allowing the addition of new problems and types of models? As a result of this systematic literature review, the present state of consistency management research is identified, along with open issues, trends and future research within this scope. Moreover, a formal approach with which to handle inconsistency problems based on transformation languages, algebraic specifications and rewriting logic [8] which will overcome the identified limitations is briefly presented.

Another survey on inconsistency management [9], which was carried out in 2001, focused on the activities of an inconsistency management process, and the positive features and limitations of the approaches that can be used in each activity. However, although the main limitations of each approach studied have also been included in our review process (see [10]), the main aim of this SLR is different since it focuses on the type of problems tackled by current work, the diagrams supported and the maintainability of each proposal, in order to answer the research question that has motivated the SLR.

The remainder of the paper is structured as follows: Section 2 describes the background and the context of this work, and the systematic literature review developed. Section 3 shows what an SLR is and describes how that which appears in this paper has been carried out. The results of the SLR are analyzed in Section 4. Finally, a preliminary approach with which to overcome the limitations found in the SLR is shown, along with our conclusions.

2. Background: model consistency concepts

Several possible definitions of model consistency and its classification appear in existing literature. This is sometimes due to the fact that these concepts are used in various, and even ambiguous or contradictory, ways within different contexts [11]. The following definitions and sources adopted for each concept related to consistency have been used in order to unify the terminology used in this paper:

- Consistency. A state in which two or more elements, which overlap in different models of the same system, have a satisfactory joint description [9].
- Inconsistency Management. A set of activities for detecting and handling consistency problems.
- Horizontal or intra-model consistency problems. These are problems between models which are built at the same level of modelling abstraction [12].
- Vertical or inter-model consistency problems. These problems can appear between models built at different levels of abstraction, for example, when refinements are made during the development [12].
- Syntactic consistency problems. This kind of consistency should guarantee that a model conforms to its abstract syntax (specified by its metamodel) [13].
- Semantic consistency problems. This consistency requires that models' behaviour be semantically compatible [13].

These definitions will be used in the remainder of the paper. The classifications between horizontal–vertical and syntactic–semantic consistency problems are orthogonal. Table 1 shows an example of each of the main classifications described (horizontal–vertical, syntactic–semantic). All the papers analyzed are included in one of these four categories.

Inconsistency problems may originate from various sources. [12] identifies two main sources of these problems:

- The multiview nature of the models: a system is described as multiple views that give the details of different concerns that make up the system, with a possible overlap between them.
- The system is developed through multiple phases or iterations, and each one produces a new, more refined description of the system.

In addition to these sources, the distributed development of a system with potentially multiple developers, which may on occasions be geographically distributed (local or global development), with multiple interpretations of both requirements and the UML notation itself [12] may also produce inconsistency problems.

3. Systematic literature review

This section shows a summary of the SLR presented in this paper. The complete report of this SLR can be found in [10].

The process of conducting a systematic literature review is not a simple task, since the whole process must be precisely defined and documented, and must include intermediate results. To facilitate the planning and execution of systematic literature reviews, [14] proposes a review protocol template based on a systematic literature review protocol previously developed in the area of medicine, along with other guidelines developed in the area of Software Engineering. In this paper, we have carried out the SLR by using the aforementioned template, and have also taken into account the guidelines given by [7]. Therefore, this template has been slightly extended to include some of the sections mentioned in [7], which were not explicitly included.

The SLR presented was carried out by effectuating the following activities:

1. Question formularization.
2. Source selection.
3. Studies selection process.
4. Information extraction.
5. Extraction execution.

The sections below give a detailed explanation of how each of these activities was carried out.

3.1. Question formularization

In this paper, and by following the instructions given in [7,14], we have defined the following research question in order to guide the SLR:

- What inconsistency problems have been tackled by the current approaches for model consistency management and how extensible are they in guaranteeing its usability and maintainability, allowing the addition of new problems and types of models?

This main question has been refined into the following set of more specific research questions (RQ):

(RQ1) What is the UML version used in the work?

Table 1
Concrete examples of consistency problem classifications.

Type of consistency	Syntactic	Semantic
Horizontal	The class names used in a sequence diagram should appear in the associated class diagram	The events produced in a sequence diagram should not produce inconsistent states in the state diagrams of the objects which participate in the interaction
Vertical	The methods definition of a class should be consistent in all the abstraction levels in which these methods could be defined	When some child classes are created in a refinement from a parent class, the traces defined by the statemachine of the parent class should be supported by the low level statemachines of the child classes

(RQ2) What is the method, language or technique used to carry out the approach (and is it a formal or a non-formal technique)? This distinction is made due to the fact that formal techniques have some features which help to avoid consistency problems such as the imprecise or ambiguous interpretation of the semantics of the modelling language [12], and also offer a wide range of applications based on their mathematical underpinning. A technique is considered to be a formal technique if it specifies at meta-level (a) a syntax, (b) semantics, and (c) a proof system [15].

(RQ3) What types of diagrams have been tackled in each approach?

(RQ4) What kind of consistency problems have been tackled in each approach?

(RQ5) How can the approach be extended to support new consistency checks? This question is related to both the usability and the maintainability of the proposal. The most up to date solutions for these problems are only partial and are of an academic nature [6], partly due to the impossibility of implementing all the possible checks by the authors of each approach. Thus, if a modeller wishes to include new consistency checks, s/he has to specify or implement them directly over the language or technique used in the approach, which may often be unknown to the modellers. The way in which each approach can be extended will therefore be considered as a further research question.

(RQ6) How suitable is the integration of the approach within CASE tools? Another feature that an approach should offer for its use within an industrial software development environment is that of good support within a CASE tool, since this helps to provide suitable feedback during modelling. In order to obtain a suitable integration between approach and tool, on the one hand, the approach should be easy for the modellers to use as a result of its integration within the CASE tool and, on the other hand, the extension mechanism offered by the approach to express new problems should also be integrated within the CASE tool. This will be considered in each work.

(a) For those approaches that do not offer integration within a CASE tool, we will consider whether they offer a prototype tool through which to facilitate the use, or learning, of the approach.

After carrying out this SLR we expect to obtain:

- Relevant information concerning the present state-of-the-art with regard to UML model consistency management.
- The identification of gaps in current research, solutions, trends and future research within this scope.
- Recommendations concerning the features that should be included in an approach which offers suitable and extensible support in the solution of consistency management problems. These will be extracted from the conclusions of the SLR.

As a consequence of these results, the definition of a novel preliminary approach for handling inconsistency problems which fulfils these recommendations is briefly presented in Section 5.

3.2. Source selection

The sources identified for use in the search for primary studies were those recommended by [7] which, from our point of view, were also appropriate for this review since they contain the work published in those journals, conferences and workshops which are of recognized quality within the research community. These sources are:

- IEEE Computer Society, search fields: title, abstract and full text;
- ACM Digital Library, search fields: title, abstract and review;
- Google Scholar, search fields: title, abstract and full text;
- Science direct, search fields: title, abstract and full text; and
- The SCOPUS Database, search fields: title, abstract and keywords.

Other important sources such as DBLP or CiteSeer were not explicitly included since they were indexed by some of the mentioned sources (e.g., Google Scholar and SCOPUS Database). In the selected sources, we experimented with various search string criteria. That which eventually retrieved the highest number of useful results was:

("management" AND "model" AND ("inconsistency" OR "consistency"))
OR ("model" AND ("inconsistency" OR "consistency"))

Certain synonyms and terms related to the concept of the model within the scope of consistency management were also taken into account in the search process. Specifically, the terms diagram, view and concern have been used as synonyms of model.

3.3. Studies selection process

Having defined the source selection, we shall now describe the process used to identify those studies that provided direct evidence with regard to the research question. We shall do this by defining a basic inclusion and exclusion criteria based on the research question, along with a procedure through which this selection was made. This is explained below.

3.3.1. Definition of inclusion and exclusion criteria

The approaches selected must be related to model consistency management. Approaches which were not based on UML models were excluded. Initially, the selection criteria were interpreted liberally and clear exclusions were only made with regard to title, abstract and introduction.

3.3.2. Procedures for studies selection

By following the indications mentioned in [7], we established a multistage process made up of three stages with different selection criteria:

- In the first stage, the search string must be run on the selected sources. An initial set of studies was obtained from the reading of the title, abstract and introduction of all the studies selected according to the inclusion and exclusion criteria. Studies which were not clearly related to any aspect of the research question were not included.
- In a second stage, the exclusion criteria were based on the following practical issues: short papers, non-English papers, non-International Conference papers and non-International Workshop papers.
- In a third stage, the selection process was based on detailed research questions (see Section 3.1).

Furthermore, and in accordance with [7], a list of studies which had been excluded as a result of the more detailed inclusion/exclusion criteria was be created. This list did not include those irrelevant papers that could be clearly excluded in the first stage after having applied the exclusion criteria.

Finally, please note that the procedure execution and all the studies selected were analyzed by the first two authors of this paper and supervised by the third.

3.3.3. Selection execution

The review process must be documented in sufficient detail [7]. The execution of our selection therefore produced various lists of studies which collected the output of each stage of the selection procedure. The information for each stage of the studies selection procedure was collected in the following manner:

- The first stage produced as output a list for each source which contained all the studies that fulfilled this first stage.
- The second stage produced as output a list of studies for each source which contained all the studies that did not fulfil the second stage inclusion criteria of the procedure for studies selection. These lists contained the excluded work together with the reason for their exclusion from the SLR.
- The third stage produced as output a list of studies for each source which contained all the studies that fulfilled the second stage of the procedure for studies selection. A completed extraction form was included for each work (see Section 3.5).

All the generated lists are available in the complete version of this SLR (see [10]).

3.4. Threats to the validity of this SLR

The main threats to validity in this systematic literature review are related to bias in the selection of the studies to be included and, in some cases, possible inaccuracy in data extraction.

We have considered the five digital libraries mentioned in Section 3.2 which, in turn, include other important electronic sources such as DBLP or CiteSeer. With regard to this point, perhaps the major validity issue facing this systematic literature review is whether we have failed to find all the relevant primary studies, although the scope of conferences and journals covered by the review is sufficiently wide for us to have achieved completeness in the field studied. Nevertheless, we are conscious that it is impossible to achieve total completeness. Some relevant papers may exist which have not been included, although the width of the review and our knowledge of this subject have led us to the conclusion that, if they do exist, there are probably not many.

The selection of papers and data extraction has been carried out by following a multistage process as is suggested in [7]. The three authors of the SLR have participated in this process, and all the selected studies have been analyzed by the firsts two authors and supervised by the third throughout the process. The criteria applied in each stage have been detailed in Section 3.3.2. It is important to note that any systematic literature review is limited to reporting the information provided in the primary studies.

Finally, in order to validate the completeness of the SLR, the studies selected in stage two were reviewed by external experts who detected some important papers that had not been included in the SLR. The primary studies in the SLR have therefore been extended with those papers [16,17] provided by experts in this field. These papers will not be taken into account in the statistics shown in either Tables 2 and 3 or in Fig. 1 since they summarize statistics related to the search process.

3.5. Information extraction

Having defined how the studies were to be selected, it was necessary to design data extraction forms to record the information obtained from the primary studies. The extraction form developed was based on the research questions defined in Section 3.1, general information related to the study identification and certain features defined in [14] related to the objective and subjective results of each study. Each paper's extraction form included the following items:

- Source. Where the paper was found (see Section 3.2).
- Study identification.
- Summary of the paper.
- Inclusion and Exclusion Criteria.
- Research Questions:
  - Method, technique or language used in the paper to manage the model consistency.

Table 2
Summary of the studies selected at each stage of the selection procedure.

	IEEE	ACM	Google Scholar	Science direct	Scopus	Total
Total results	10	170	485	24	218	907
Results selected (stage one)	10	12	24	7	37	90
Results selected (stage two)	4	6	12	5	27	54

Table 3
Summary of search engines overlap in the second stage.

	IEEE	ACM	Google Scholar	Science direct	Scopus
IEEE		[18]	[19]	0	[18,20]
ACM	[18]		[21,22]	0	[20,18,21,23,22]
Google Scholar	[19]	[21,22]		[24]	[18,21,22,24]
Science direct	0	0	[24]		[18,24,25]
Scopus	[18,20]	[20,18,21,23,22]	[18,21,22,24]	[18,24,25]	

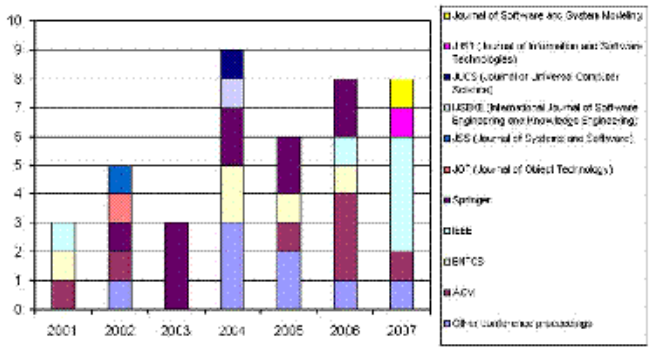

Fig. 1. Number of publications over specific time period.

- UML version. If the approach used a simplified or owned version of the UML metamodel, this field was filled in with "Simple UML".
- Formal approach (Yes/No).
- Diagram support. Types of diagrams supported by the approach.
- Consistency support: Syntactic–Semantic. Horizontal–Vertical.
- Extension mechanism.
- CASE tool integration.
- Automatic support.

– Objective Result Extraction
- Study Results: effect obtained through the study review.
- Study Problems: study limitations found by the paper's author.

Table 4
Summary: technique and support.

Reference	Technique	UML version	Formal	Extension mechanism	CASE Integration	Automatic support
Amaya et al. [26]	xLinkit	Simple UML	NO	NO	NO	NO
Amalio et al. [27]	Z	Simple UML	YES	NO	NO	NO
Chiorean et al. [24]	OCL	1.3	NO	OCL	NO	YES, OCLE
Diethers et al. [28]	Timed Automata (UPPAAL model checker)	1.3	YES	NO	YES	YES
Egyed et al. [20,18,23]	Constraints defined with the language/checker available in IBM rational Rose	1.3	NO	NO	YES (IBM Rational Rose)	YES
Egyed et al. [29]	Transformation and comparison	1.3	NO	NO	YES (IBM Rational Rose)	YES
Engels et al. [30,31,25,32–34]	CSP (Communication Sequential Process)	1.3	YES	NO	NO	YES, consistency workbench
Fryz et al. [35]	Graphs	2.0	YES	NO	NO	NO
Graf et al. [36]	Manual inspection	1.4	NO	NO	NO	NO
Hausmann et al. [37]	Graphs	1.4	YES	NO	NO	NO
van Hee et al. [38]	Petri Nets (PN) and Z	Simple UML	YES	NO	NO	NO
Inverardi et al. [39]	SPIN model checker	Simple UML	YES	NO	NO	YES
Khalkar et al. [40]	SAL (Symbolic Analysis Laboratory)	Simple UML	YES	NO	NO	YES
Laleu et al. [41]	B	Simple UML	YES	NO	NO	YES
Lam et al. [42]	pi Calculus	2.0	YES	NO	NO	NO
Lucas et al. [43]	Rewriting Logic (Maude)	Simple UML	YES	NO	NO	NO
Mulgaonkar et al. [22]	CLP (Constraint Logic Programming)	Simple UML	YES	NO	NO	NO
Mens et al. [44]	Graph transformation rules	Simple UML	YES	NO	NO	YES, AGG
Ossami et al. [45]	B	Simple UML	YES	NO	NO	YES, B/Smurf tools
Paige et al. [21,46]	PVS (theorem proving), Eiffel and JML (OO programming)	Simple UML (BON diagrams)	YES	NO	NO	YES, BON CASE
Paige et al. [47]	OCL	Simple UML	NO	OCL	NO	NO
Rasch et al. [48]	Object Z and CSP	1.5	YES	NO	NO	YES, FDR
Sapna et al. [19]	SQL triggers	Simple UML	NO	OCL	NO	NO
Schrefl et al. [17]	Petri Nets	Simple UML	YES	NO	NO	NO
Shinkawa et al. [49]	Colored Petri Nets (CPN)	Simple UML	YES	NO	NO	NO
Spanoudakis et al. [50,51]	OCL	Simple UML	NO	OCL	NO	YES, S tool
Straeten et al. [52,53]	Description Logic (DL)	2.0	YES	NO	YES	YES
Wagner et al. [16]	Graphs	1.5	YES	NO	YES, plugin of Fujaba	NO
Wang et al. [56]	FSP (Finite Transition System)	Simple UML	YES	NO	NO	YES
Yu et al. [57]	Petri Net (PN)	Simple UML	YES	NO	NO	YES, Owner tool
Yeung et al. [58]	B and CSP	1.4	YES	NO	NO	NO
Zhao et al. [59]	SPIN model checker	Simple UML	YES	NO	NO	YES

- Subjective Results Extraction
  - General Impressions and Abstractions: subjective conclusions after studying the paper.

This form was used for classifying each study. A complete list of the forms filled in for each primary study can be found in [10]. Part of this information is included in the following sections.

3.6. Extraction execution

The systematic literature review took place in March 2008 and no limitations were imposed on the years covered by the search (although the studies which were eventually analyzed were from 2001 to 2007). Table 2 shows a summary of the studies selected in each stage of the selection procedure for each source. The "Total results" are those results which were obtained by running the search string on the selected sources. The next two rows show the results obtained after applying stages one and two of the studies selection procedure. The approaches resulting from this last stage were studied in depth and information concerning the detailed research questions and other fields of the extraction forms was extracted from each paper. Although 56 works were selected, some of them appeared in different sources, so repeated studies were eliminated. 44 works were eventually analyzed (see first column Table 4). Moreover, Fig. 3 summarizes the overlapping search engines produced in the second stage, in order to provide details of how many papers were found in various different sources. Fig. 1 presents an overview of the number of selected publications over a specific period of time, together with their publishers. As the reader can notice, consistency management has attracted the attention of numerous quality journals and conferences in recent years, which denotes that this still is a very active field of research.

4. Results of the systematic literature review

This section shows a summary of the results obtained in the SLR, along with an analysis of the data collected, in order to identify open issues, limitations of the current approaches and recommendations which can be used to offer a suitable model consistency management within the scope of UML.

Once the extraction execution had been completed, it was important to ensure that multiple publications of the same approach were not included in the data analysis. Papers that appeared in multiple sources were thus taken into account only once. Once the duplicated papers had been removed, the different papers describing the same data or approach were grouped together, since duplicate reports would seriously bias the results of the data synthesis [7]. The total number of different approaches included in Tables 4 and 5 is 32.

The results are summarized in two tables. Each of the table's entries groups the papers according to the author's own approach, and these entries have been ordered by the year of publication and the first author's name. Table 4 shows the values regarding to the research questions related to "Technique", "UML version", "Formal", "Extension mechanism", "CASE integration" and "Automatic support". This table therefore shows the general information of each approach, i.e., which method is used and what the tool support offered by each approach is.

Table 5 shows the diagrams supported by each approach and what kind of inconsistency problems they handle. The remaining items included in the extraction form for each approach can be found in [10].

Moreover, Fig. 2 and Table 6 show schematically which diagrams and what consistency support is offered in current literature. Fig. 2 shows the percentage of diagrams involved in the proposals. As we can see, three diagrams (class, state and interac-

Table 5
Summary: diagram and consistency support.

Reference	Diagram support	Consistency support
Amaya et al. [26]	Use cases, class and sequence	Semantic Horizontal
Amalio et al. [27]	Class and statecharts	Syntactic/Semantic Horizontal
Chiorean et al. [24]	Class	Syntactic Horizontal
Diethers et al. [28]	Sequence and statecharts diagrams	Semantic Horizontal
Egyed et al. [20,18,23]	Class, sequence and statecharts	Syntactic Horizontal
Egyed et al. [29]	Class	Syntactic Vertical
Engels et al. [30,31,29,32–34]	State diagrams	Semantic Horizontal/Vertical
Fryz et al. [35]	Use cases and class	Syntactic Horizontal
Graaf et al. [36]	Sequence and statecharts	Semantic Horizontal
Hausmann et al. [37]	Class and objects	Syntactic Horizontal
Hee et al. [38]	Class, sequence and statecharts	Syntactic Horizontal
Inverardi et al. [39]	Statecharts and sequence	Semantic Horizontal
Khalkar et al. [40]	Use cases and class + owner diagrams (see [10] for details)	Semantic Horizontal
Laleau et al. [41]	Class, statecharts and communication	Syntactic Horizontal
Lam et al. [42]	Sequence and statecharts	Semantic Horizontal
Lucas et al. [43]	Class and communication	Syntactic Horizontal
Malgouyres et al. [22]	Class	Syntactic Horizontal
Mens et al. [44]	Class and statecharts	Syntactic Vertical
Ossami et al. [45]	Class and statecharts	Syntactic/Semantic Horizontal/Vertical
Paige et al. [21,46]	Class and communication	Syntactic/Semantic Horizontal/Vertical
Paige et al. [47]	Class and sequence	Syntactic Horizontal
Rasch et al. [48]	Class and statecharts	Semantic Horizontal
Sapna et al. [19]	Use case, activity, class, sequence and statecharts diagrams	Syntactic Vertical
Schrefl et al. [17]	Statecharts	Semantic Horizontal/Vertical
Shinkawa et al. [49]	Use case, sequence, activity and statecharts	Syntactic Horizontal
Spanoudakis et al. [30,31]	Class and sequence	Syntactic Horizontal
Straeten et al. [32–35]	Class, sequence and statecharts	Syntactic/Semantic Horizontal/Vertical
Wagner et al. [16]	Class	Syntactic Horizontal
Wang et al. [36]	Sequence and statecharts	Semantic Horizontal
Yao et al. [37]	Sequence and statecharts	Semantic Horizontal
Yeung et al. [38]	Class and statecharts	Syntactic/Semantic Horizontal
Zhao et al. [39]	Sequence and statecharts	Semantic Horizontal

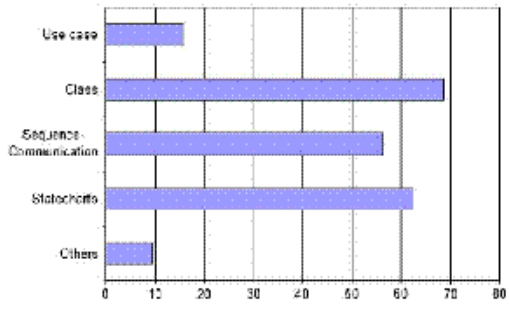

Fig. 2. Percentages of diagram support.

Table 6
Percentages of consistency support.

Type of consistency	Syntactic	Semantic
Horizontal	53.13% (17 of 32)	53.13% (17 of 32)
Vertical	18.75% (6 of 32)	15.63% (5 of 32)

tion diagrams) are tackled by more than 60% of the proposals, whereas the remaining UML diagrams are usually overlooked.

Table 6 shows the percentage of approaches tackling each type of consistency. The table clearly demonstrates that vertical inconsistency problems are studied with less frequency than those which are horizontal.

4.1. Analysis of results, conclusions and recommendations

Having shown the results of the systematic literature review, in this section we provide: (1) our conclusions, which were obtained by analyzing the collected data for each research question and our own experience; and (2) recommendations for possible future research based on the research questions used in this review.

4.1.1. UML version

The majority of the approaches presented (60%) use simplified versions of the UML metamodel. This is a disadvantage with regard to the use of the approaches in industrial software development, but this problem is understandable since many authors do not have the support of companies and their proposals are of an academic nature and solely attempt to fill a gap that exists in current UML modelling. The other proposals (40%) use the UML version available at the time of their publication or the version offered by the tool used for building the models.

Although this is an understandable limitation, it is one that should be rectified in future approaches. A means of solving this might be through the integration of the proposal into a standard software development platform such as EMF (Eclipse Modelling Framework) [60], which allows different metamodels (UML 2.0, simple metamodels, DSLs (Domain Specific Languages), ...) to be worked on since it includes an implementation of EMOF [61].

4.1.2. Formal approach

75% of the techniques used in the proposals for detecting and handling inconsistency problems are formal. None of the formal techniques used stand out above the others, although almost all the proposals that tackle semantic inconsistency problems use a model checker to simulate the behaviour of the system and to find inconsistencies.

This high percentage reveals that the use of formal techniques offers advantages in dealing with consistency problems. Formal techniques add precision to UML models, and their mathematical underpinning permits the use of a wide range of applications such as model checkers, theorem provers, coherence checkers, etc., to name but a few. Many approaches therefore use these methods, in spite of the fact that formal techniques are not yet very popular in the industrial software development community [62]. This unpopularity is usually due to the fact that these approaches are difficult for the modellers to use directly, and that the feedback they offer is usually poor and difficult for non-experts to understand. However, this problem can be solved if the approach is integrated into standard software development platforms [62].

Table 7
Specification technique paradigm.

Reference	Formal technique	Paradigm
Amalio et al. [27]	Z	State transitions
Diethers et al. [28]	Timed Automata (UPPAAL model checker)	State transitions
Engels et al. [30,31,29,32–34]	CSP (Communication Sequential Process)	Process algebra
Hausmann et al. [37]	Graphs	State transitions
Fryz et al. [35]	Graphs	State transitions
Heeren et al. [38]	Petri Nets (PN) and Z	State transitions
Inverardi et al. [39]	SPIN model checker	Logic
Khalkar et al. [40]	SAL (Symbolic Analysis Laboratory)	State transitions
Laleu et al. [41]	B	State transitions
Lam et al. [42]	pi Calculus	Process algebra
Lucas et al. [43]	Rewriting Logic (Maude)	State transitions
Mulgouyres et al. [22]	CLP (Constraint Logic Programming)	Logic
Mens et al. [44]	Graph transformation rules	State transitions
Ossami et al. [45]	B	State transitions
Paige et al. [21,46]	PVS (theorem proving)	Logic
Rasch et al. [48]	Object Z and CSP	State transitions and Process algebra
Schrefl et al. [17]	Petri Nets	State transitions
Shinkawa et al. [49]	Colored Petri Nets (CPN)	State transitions
Straeten et al. [52–55]	Description Logic (DL)	Logic
Wagner et al. [16]	Graphs	State transitions
Wang et al. [56]	FSP (Finite Transition System)	State transitions
Yao et al. [57]	Petri Net (PN)	State transitions
Yeung et al. [58]	B and CSP	State transitions and Process algebra
Zhao et al. [59]	SPIN model checker	Logic

Due to the advantages offered by these methods, we recommend that inconsistency problems be tackled with the use of formal techniques, although a suitable support within a CASE tool should be developed if they are to be used in industrial software development.

Finally, in order to show what the most frequently used formal techniques are, we have classified each technique according to the four paradigms identified in [63]:

- State transitions: the specification describes a transition relation on a set of *states*, e.g. B, Z, Petri Nets, etc.
- Algebra: the specification describes a set of operations defined on a set of types (also called *sorts*). Events are represented by a function (also called an *operation*). The behaviour of functions is given by a set of equations (axioms) which states how functions are related, e.g. CASL.
- Process algebra: this is a special type of algebra. Its operations are applied to elementary processes and events to describe how events may occur, e.g. (e-)LOTOS.
- Logic: the behaviour of functions is given by a set of equations (axioms) which states how functions are related, e.g. PVS, pi-calculus, etc.

More detailed information about these definitions can be found in [63].

In Table 7, each approach has been classified according to this classification. Fig. 3 shows a summary of this information. As the table shows, most of the approaches presented (71%) use a state transitions technique.

4.1.3. Diagram support

The diagrams tackled by each approach alter according to the kind of consistency problem that they wish to check or handle. For example, approaches that deal with behaviour consistency problems usually consider sequence and state diagrams, whereas approaches for static consistency problems usually tackle class diagrams together with other diagrams that use the information defined in the class diagram such as sequence, communication or state diagrams.

As Fig. 2 shows, in general, the interaction (sequence or communication), statecharts and class diagrams are those most frequently tackled, whereas other UML 2.0 diagrams such as timing, activity or components diagrams are not studied. Although this may be a limitation, studies such as [64] demonstrate that these diagrams are those which are most frequently used in industrial software development. The diagrams checked are therefore sufficiently representative to prove the validity of a proposal.

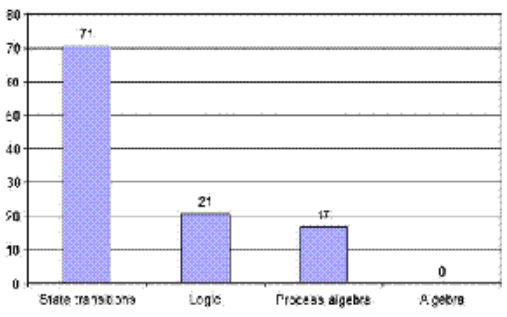

Fig. 3. Summary of formal paradigm.

4.1.4. Consistency support

Table 6 shows a summary of the consistency support offered by the selected proposals. Note that the consistency problems most frequently tackled are syntactic and horizontal problems.

One of the most important conclusions that may be reached by studying this figure is that only a small number of proposals cover problems related to vertical consistency. This kind of problems is particularly important within the scope of MDE approaches in which consistency problems at different levels of abstraction arise. Future proposals dealing with the solution of consistency problems should, therefore, consider this type of inconsistencies.

4.1.5. Extension mechanism

As was previously mentioned, the extensibility of a proposal is related to its usability and its maintainability. However, Table 4 shows that only 12.5% of the approaches offer a means of extending the proposals without being forced to program in the technique used. These proposals use OCL (Object Constraint Language) [65] as a language through which to express consistency problems. However, OCL is too limited to express them. For example, it does not allow us to fix any inconsistencies found since its constraints are side-effect free and cannot modify model elements. Other problems related to OCL, such as the fact that it is not easy to understand, are identified in [66].

On the other hand, as was mentioned in the previous section, the remaining approaches (87.5%) are limited to the implementation offered by the authors, since it is impossible to include all the possible inconsistency checks made by the authors of the approach. This limitation makes expert support necessary if support for new UML diagrams or consistency checks and handling are to be included. This limitation becomes more meaningful within the scope of DSL development, which is rising due to its use in the majority of the proposals connected with MDE. New kinds of models are defined within this scope, so the lack of maintainability also appears when we wish to tackle inconsistency problems within these new models.

This limitation signifies that the solutions proposed for these problems are only partial. However, they could be overcome if the approach were to offer a way in which to extend them through a well-known language, which has sufficient power to express both the definition and the handling of consistency rules or constraints and expressiveness, thus making them easy to read and understand.

4.1.6. CASE tool integration

The studied approaches are usually supported by a tool (53.1%), but only 15.6% of them are suitably integrated within a CASE tool which permits the easy use of the checks implemented. Moreover, the non-integrated approaches in a CASE tool offer poor feedback which is difficult for modellers to understand, since the output of the consistency checks is usually expressed within the scope of the technique chosen (for example, the formal technique used).

Future new approaches should not only facilitate the use of the checks already implemented, but also integrate the definition of new consistency checks and handlings through to the extension mechanism and improve the feedback that the consistency check produces with the aim of easing the modellers task of identifying and handling the problems detected in the models. To do this it will be necessary to select and extend one of the existing CASE tools. The extension capacities of CASE tools should, therefore, be taken into account during the selection process.

4.2. Other future research trends

In addition to the observations concerning each research question which may serve to guide future research in model consis-

tency management, this SLR has also brought to light other considerations, recommendations and open issues which should be taken into account.

4.2.1. Inconsistencies in MDE

Model transformations have always been an important field of research through which to automate (totally or partially) model evolution throughout downstream development, or simply to discover models which improve certain features of the source models (based on certain criteria: experience, metrics...). Furthermore, in recent years, the profound impact of the MDE proposal has meant that model transformation has become a highly active direction for research and development. Since MDE proposes the construction of a system's models at different levels of abstraction, from models that do not contain details of a specific platform to models that take into account the features of the specific platform in which the system will be implemented, inconsistencies in the more abstract models could make (semi-)automatic generation impossible [6]. Moreover, changes in less abstract models may produce inconsistencies with regard to their more abstract models. Vertical consistency therefore becomes one of the most relevant unresolved problem since, as we have already mentioned, it is usually overlooked in the approaches used to handle inconsistency problems.

4.2.2. Handling inconsistencies

A concrete inconsistency problem can be handled by several solutions. Moreover, in most situations, the modeller should decide which solution is the most suitable for the system that is being developed. For instance, if a method used in a sequence diagram does not appear in its corresponding class diagram it is possible to, for example, add a new method to the class of the object that receives the message or modify the method of the message with one from that class. Although, some approaches handle the inconsistencies found, this handling might not be the most suitable for the system, so new approaches should offer the possibility of defining and choosing among multiple possible actions.

5. Our proposal: model consistency management powered by transformation languages

Having taken into consideration the recommendations for surveys provided in [67], in this section we define an approach which attempts to show the viability of overcoming the limitations and unresolved situations identified by the systematic literature review. The features of this approach are focused on offering an approach that will be: (1) extensible, (2) aligned with the MDA proposal, and (3) suitably CASE tool integrated. By taking these aims as our starting point, this section proposes an initial approach based on three elements:

- Transformation languages, which will be the basis for the definition of the extension mechanism and the front-end of the proposal.
- Rewriting logic [8]. Owing to the benefits of formal techniques already commented on, the approach will be implemented over a formal language in order to use its mathematical underpinning to enable theoretical properties to be proved. The formal language used to give support to the whole proposal is Maude [68]. This technique will be the back end of the proposal.
- A CASE tool which appropriately integrates all these technologies based on Eclipse EMF [60].

Since the approach presented is not the main aim of this paper and will be presented in further work, these features will be explained only briefly in the following sections.

5.1. A transformation language for model consistency management

One of the most important limitations of the current approaches used outside academic scopes is the absence of an extension mechanism. This problem can be solved by offering a well-known intermediate representation or language to extend the approach.

Since a consistency problem can be seen as a set of relationships that must be held among model elements conforming to one or several metamodels, and since transformation languages define relationships among the metamodel elements that will be transformed, our proposal consists of reinterpreting the semantics of one of these transformation languages and using it as a basis to define an extensible approach for model consistency management.

Several transformation languages with which to define transformations have appeared within the scope of MDE [69–71]. These definitions are usually expressed as a set of transformation rules that describe how a source model is transformed into a target model [72]. That is, these rules describe how metamodel elements of the source model are transformed into other metamodel elements in the target model. From our point of view, the semantics of these transformation rules will be reinterpreted in order to express relationships that must be held among metamodels, i.e., consistency relationships that must exist among metamodels. An inconsistency problem will thus be expressed as a set of relationships (by means of transformation rules). With these semantics, it would appear to be suitable to use a transformation language such as an intermediate language which is well-known to modellers.

The language chosen for our approach is that of QVT Relations [69], proposed by the OMG (Object Management Group) within the scope of MDA. In this language, relations among metamodels establish how the transformations are carried out. The main reasons for choosing this language as the basis of our approach have been:

- It is one of the languages defined by the OMG in QVT. This guarantees a wide acceptance within the software development community.
- *QVT Relations* is the most abstract and user-friendly language of all the languages defined within the QVT standard [69].
- It is capable of expressing any kind of transformation, and thus any kind of consistency problem among metamodels.

Therefore, QVT Relations provides a well-known and sufficiently expressive language which, when interpreted in this way, provides a mechanism through which to define consistency relationships in industrial software development. Other features of this language are declarative specification and complex object pattern matching, features which are easily supported by our approach.

5.2. The formal language maude

The formal language chosen to specify our approach is Maude [68]. This language is based on equational and rewriting logic and its specifications are executable, thus allowing us to build prototypes, check constraints over a system, and prove theoretical properties over the behaviour of a system in an efficient manner (from half a million to several million rewrites per second [68]).

In rewriting logic [8], a system is specified through a rewrite theory, which consists of a signature Σ (sorts and operations), a set E of equations, and a set of rewriting rules. The static part of a system is modelled by means of equational logic (Σ and E), and the dynamic part is specified by adding rewriting rules, that is, a set of rules which specify how the system's state changes.

One of the most important concepts of rewriting logic that will be used in our approach is the concept of the rewriting rule. A rewriting rule (named l) describes a local concurrent transition that can take place in a system. If the pattern on the left-hand side of the rule (t) matches a fragment of the system state, the matched fragment is transformed into the corresponding state of the right-hand side of the rule (t'), which is expressed as: $l : t \rightarrow t'$.

Maude allows both the specification of this kind of logic and its execution in an object oriented manner, in which the system elements are represented as classes that can be instantiated as objects that exchange messages between each other. Maude also offers several extension modules which can be used in the context of model consistency management. Two of these are the Maude strategies language, which allows us to establish the execution order of the consistency rules, and the Maude metaprogramming capacities, which help us to link rewriting logic with the front end of the proposal.

5.3. Model consistency management through IQVT-Maude language

Fig. 4 summarizes the main elements of which the approach is made up. This figure shows OMG standard elements, the Maude elements, the relationships among them and how an inconsistency problem in natural language will be expressed, firstly as QVT Relations and then as rewriting rules, in the formal back end. The role of each element in this approach will be shown in the following sections by means of an example.

5.3.1. Metamodels specification

The metamodels used in this paper will be specified according to the concepts given in Section 5.2. Metamodel elements will be specified in Maude by means of classes with attributes that describe the metamodels, and their instances (*Maude objects*) represent models that conform to their corresponding metamodel. These relationships are shown in Fig. 4 by means of solid lines between models and metamodels.

In this Figure, two metamodels are involved in the consistency definition. However, in general, more than two metamodels can appear, and the definition may even involve only one metamodel. As an example, in this work we will use metamodels of a simple UML class diagram and a simple UML sequence diagram. Both will be specified in Maude, and their models will be instantiated through *Maude objects*. These metamodels are shown in Fig. 5.

5.3.2. QVT Relations features in maude

In this subsection, we briefly analyze the basis of QVT Relations and how they are also presented in Maude. This concept is summarized in Fig. 6.

QVT Relations is a declarative model transformation language, which means that its implementation using a declarative language such as Maude is more "natural" than other non-declarative languages.

On the one hand, a relation declares constraints that must be satisfied by the metamodels (or domains) that participate in the relation. Each domain establishes a pattern (with a set of variables and constraints) that must be matched with the candidate models in order for the transformation to be carried out. These are known as *object template expressions*, which are directly expressed in Maude since pattern-matching is one of its features.

These QVT Relations will be specified as Maude rewriting rules (see Fig. 4) which change, create or, in this case, verify the elements in the models. Moreover, the constraints express conditions over the models which will be specified as conditions in the rewriting rules.

QVT Relations provide all these elements textually, but once they have been specified in Maude, they are transformed into mathematical entities, thus enabling us to take advantage of the total power of mathematical inference mechanisms, without losing the intuition of the QVT concepts.

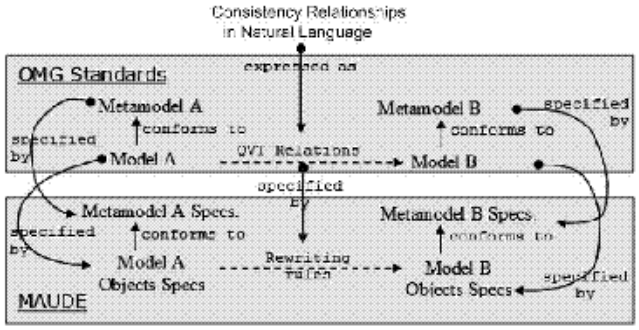

Fig. 4. Summary of the approach elements.

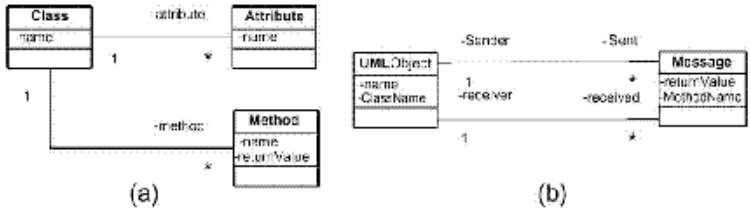

Fig. 5. Simple class diagram (a) and simple sequence diagram (b) metamodels.

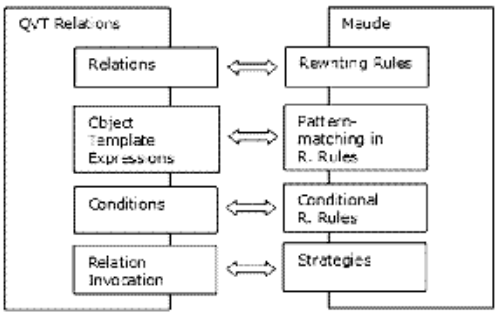

Fig. 6. Equivalence of concepts' features.

```
relation SequenceObjectMessageToClassMethod{
 cn, mn, rv:String;
 checkonly domain sequence o:UMLObject{
  className = cn,
  received = ms:Message {
   methodName = mn, returnValue = rv
  }
 };
 enforce domain class c:Class{
  name = cn,
  method = mt:Method { name = mn, returnValue = rv }
 };
}
```

Fig. 7. Consistency relation in the use of methods through a QVT relation.

5.3.3. IQVT-Maude language

The main features of QVT Relations can be supported directly in Maude. However, the pattern–matching of Maude is based on the syntax which defines the terms of the model elements, so the means of expressing relations between objects and matching them during their execution is slightly different to that offered by QVT Relations, which would not allow Maude to be directly usable by modellers. In order to solve this problem, we have used Maude's metaprogramming capacities to extend it with a new pattern–matching language called the IQVT-*Maude* (*like-QVT-Maude*) language, which will be used to join the front end of the proposal with the back end.

In order to implement this pattern language based on QVT Relations, Maude also allows us to define the grammar of domain–specific languages (in our case, a simplified version of the QVT Relations pattern–matching grammar, named IQVT-*Maude*). Since the pattern matching and its execution in Maude are based on the Maude grammar, this specific language is not directly executable in Maude. To solve this problem, once the grammar has been defined, Maude allows the specification of *metalanguage* applications, in which Maude parses and handles the new language using its metaprogramming capacities. In our approach we will write rewriting rules, using the IQVT-*Maude* language to define the patterns on the left-hand and right–hand sides of the rules, and metaprogramming is used to handle each rule and to transform it into an executable patterns based on the Maude grammar.

As the following sections will show, rewriting rules and a like-QVT pattern–matching will be used as an extension mechanism for the approach for managing consistency constraints.

5.3.4. Example: consistency between class and sequence diagrams

This section shows (1) how the QVT Relations language can be used to define and handle consistency relationships, and (2) how IQVT-*Maude* is used to express these QVT relations in Maude. To do this, we define an example of a consistency relationship in the use of methods between Sequence and Class Diagrams. In this particular case, a consistent use of a method in the sequence diagram ("methodName") implies that a method exists with the same name in the class of the object ("className") which receives the message, and that the type of the return value is also the same. Fig. 7 shows the QVT Relation that expresses this consistency relationship.

Two domains are involved in this relation: a sequence model and a class model. The rule establishes the relation that must exist among the elements of these models. To do this, it matches objects in each domain that hold the relation. The variables "cn", "mn" and "rv" are used to match the objects of the models. Fig. 8 shows a schema of the objects that hold the relation. If it is not possible to find a set of objects that match the relation, an inconsistency problem exists.

Once the relation has been defined, it will be expressed in Maude through IQVT-*Maude* and rewriting rules. The purpose of the rewriting rules will be firstly to check the consistency relationship and, if this fails, to handle the inconsistency problem. As will be observed, the rewriting rules using IQVT-*Maude* will still be syntactically different from the relation in Fig. 7, since certain implementation details must be considered in the current specification of IQVT-*Maude*. Moreover, the concept of the rewriting rule does not exist within the scope of QVT, but is close to the transformation

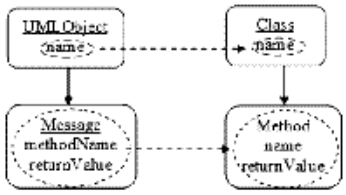

Fig. 8. Schema of the objects matched in the relation.

```
vars cn mn rv : String .
rl [SequenceObjectMessageToClassMethodCheck] :
 domain "sequence" o:UMLObject{
  className = cn,
  *** The "checked" attribute is used to find
  *** messages that have not been checked yet.
  received=ms:Message {
   methodName = mn, returnValue = rv,
   checked = false }
 }
 domain "class" c:Class{ name = cn,
  method=mt:Method { name = mn, returnValue = rv }
 }
=>
 domain "sequence" o:UMLObject{
  className = cn,
  *** If a message holds the consistency relation,
  *** it is marked as checked.
  received=ms:Message {
   methodName = mn, returnValue = rv,
   checked = true }
 }
 domain "class" c:Class{ name = cn,
  method=mt:Method { name = mn, returnValue = rv }
 } .
```

Fig. 9. IQVT-*Maude* rule to check the inconsistency problem.

rule definition, and the use of IQVT-*Maude* brings its syntax closer to that of the QVT Relations.

Fig. 9 shows the IQVT-*Maude* rewriting rule, called "SequenceObjectMessageToClassMethodCheck". The left-hand side of this rule matches the objects that fulfil the consistency relation and, once they have been found, the right-hand side of the rule marks the element that must be consistent (the message in this example) as being correct. The extra attribute "checked" (which does not appear in the QVT relation) is used to mark the elements, and sequence diagrams that do not hold the relation will therefore contain unmarked messages after checking has taken place. This is one of the implementation details that must be taken into account in the current implementation of IQVT-*Maude*.

Other implementation details that make the rewriting rule in Fig. 9 different from the relation in Fig. 7 are the definition of variables by following the Maude grammar, certain syntactical differences in the domain definitions or the need to include the left-hand and right-hand sides in the rewriting rule. These differences signify that, future versions of IQVT-*Maude* will be extended to offer a syntax which is closer to the QVT Relations language.

Once the inconsistency has been found, it is necessary to define how to handle it. However, as was mentioned in Section 4.1, a concrete inconsistency may have several solutions. IQVT-*Maude* and rewriting rules are therefore used to allow modellers to define new ways of handling inconsistency problems. In this example, two possible solutions to this inconsistency problem are: to add a new method to the object class that receives the message or to find a candidate method in the class and change it in the message in order to rectify the inconsistency. The specification of only the first solution is shown due to lack of space. Fig. 10 shows the rule which adds the lost element to the source model in order to rectify the inconsistency problem. The left-hand side of the rule shows a pattern that matches the necessary model elements to rectify the inconsistency (in this case, the object and the message in the sequence model, and the class of the object in the class model). The right-hand side of the rule is used to rectify the inconsistency by adding a new element (a method in this example) to the class model. As Fig. 10 shows, the attribute "checked" is used in the left-hand side of the rule to find inconsistent messages previously checked.

```
vars cn mn rv : String .
*** Once a model has been checked, we can fix the
*** errors found.
rl [SequenceObjectMessageToClassMethod] :
 domain "sequence" o:UMLObject{
  className = cn,
  *** The value of the "checked" attribute is false
  *** for the messages that do not hold the
  *** consistency relation.
  received=ms:Message {
    methodName = mn, returnValue = rv,
    checked = false }
 }
 domain "class" c:Class{
  name = cn, method = mt:Method{}
 }
=>
 domain "sequence" o:UMLObject{
  className = cn,
  *** Once the inconsistency has been fixed, the
  *** value of the "checked" attribute is
  *** established to true.
  received=ms:Message {
    methodName = mn, returnValue = rv,
    checked = true }
 }
 domain "class" c:Class{
  name = cn,
  *** A new method is created in the class named "cn".
  method=mt:Method{ name = mn, returnValue = rv,
    checked = false }
 }
```

Fig. 10. IQVT *Maude* rule to handle the inconsistency problem.

Modellers can therefore use IQVT-Maude and rewriting rules to write the rules which are necessary to manage the consistency problems of their models. The operations over these rules (definition, modification, choosing a specific checking and handling ...) will be offered and managed through the CASE tool integration currently being developed (see the following section).

5.4. A glance at managing semantic inconsistency problems

Finally, this section briefly analyzes how the semantic inconsistency problems can be defined and tackled by only defining transformation rules, in order to show how the presented approach can be also used to manage this kind of problems.

5.4.1. Behaviour definition

Semantic inconsistency problems are related to the behaviour defined by the semantics of the different models (metamodels) involved in the system development. From our point of view, transformation rules (IQVT-*Maude* + rewriting rules) will be used firstly to express how the system behaviour, represented by models, changes. In general, this definition may involve any number of metamodels. For example, if we define an example of a consistency relationship between statemachine and sequence models, we can define their behaviour based on a simulation of the statemachine of each object that appears in the sequence diagram using the received messages as its lifeline. Each message received by an object will thus provoke a change in the state of the statemachine.

5.4.2. Consistency checking

Once the behaviour has been defined, Maude inference mechanisms and commands are used to execute the transformation rules and check these problems. Note that, since different semantic inconsistency problems can be defined over the same behaviour defined, the same behaviour definition can be used to manage all of them. In this example, we can define many possible semantic inconsistency problems between statemachine and sequence models. One of them is to check the consistent behaviour between an object that takes part in the interaction and its associated statemachine. That is, if a sequence of messages received by an object is not supported by its statemachine, an inconsistency problem exists. This problem can be checked by means of the Maude rewriting command, which allows us to use the previous behaviour to simulate a statemachine. This simulation would have two possible terminations: (1) the sequence of messages is fully simulated (meaning that the statemachine and the sequence diagrams are consistent); or (2) a message of the sequence can not be simulated in the statemachine since a transition fired with that message does not exist.

5.4.3. Consistency handling

Once the inconsistency has been found, in our approach model transformations are again used to define how to handle it. In this example, once the inconsistency has been found, several possible solutions to this problem exist. In our case, when an inconsistency is found, the transformation rule will add a new state and a new transition to allow the statemachine to tackle the inconsistent message (this process was used in [73] to create partial statemachine models from sequence models). Note that the next messages to appear in the sequence diagrams will become the new inconsis-

tency problems, since new outgoing transitions in the new created state do not exist, so the checking–handling process will have to be repeated for the remaining messages in the interaction to allow the complete sequence of messages to be tackled by the statemachine model.

Finally, note that if the proposed solution is not useful, the modeller can only check the models and fix them by manual handling or by defining her/his own transformation rule for handling it.

5.4.4. Other *maude commands* for tackling semantic inconsistency problems

Another Maude inference mechanism could be used to check other inconsistency problems. For example, some Maude tools allow us to check deadlock among any set of rules. Since Maude rewriting rules are used to define the models behaviour, we can use these tools to check deadlock free behaviours. We can therefore, for example, define the behaviour of a statemachine (as in Section 5.4.2) and check that the use of several statemachines for one system is consistent in terms of deadlocks and livelocks.

Thanks to the chosen formalism, the proposed approach thus offers the possibility of tackling both syntactic and semantic problems while most of the approaches studied tackle only one kind since their techniques are more sensitive to each kind of problem. The use of this proposal in semantic consistency will be presented in greater depth in future work.

5.5. CASE tool integration: eclipse EMF

In general, a quite common disadvantage of a formal approach is related to the lack of a suitable and transparent integration with the tools used by the modellers during the development life cycle.

We have chosen the EMF [60] for our implementation. This has become one of the most frequently used modelling tools in academic and industrial environments [74,75] due to its plug-ins mechanism. This mechanism makes it easy to reuse and to add functionality. Another of the most important advantages of using EMF is the full and easy access that it offers to its model and metamodel representation, which is essential for tackling the implementation of our proposal.

With regard to our approach, we use the Ecore metamodel of EMF both to support UML 2.x, and to define new metamodels and models. Moreover, we use MOFScript [76] to transform these metamodels and models into code (in this case, Maude specifications from metamodels and objects from specific models). The Eclipse plug-in mechanism is useful in that it offers IQVT-*Maude* language directly in the EMF environment and a suitable integration of all the elements involved in the approach. The main features that this environment offers are: (1) the possibility of selecting the inconsistency checks that will be executed, (2) the possibility of selecting a solution for inconsistency handling once a problem has been detected, (3) the definition of new consistency problems and how to rectify them, and (4) suitable feedback to the modeller with regard to the activities executed. This integration is still under development and will be shown in its entirety in future work.

Table 8
Answers for the research questions of the proposal.

Research question	RQ value		Technique used
UML version	2.x		Eclipse EMF
Formal approach	YES		Maude
Diagram support	UML 2.x, DSLs,...		Eclipse EMF
Consistency support	Syntactic/Semantic Vertical	Horizontal/	IQVT-Maude rewriting rules
Extension mechanisms	YES		QVT Relations
CASE tool integration	YES		Eclipse EMF
Automatic Support	YES		Eclipse

5.6. Research questions for the proposal

This section summarizes how the different research questions defined in the SLR are answered in the preliminary proposal presented (see Table 8). All the features of the proposal and its use for tackling inconsistency problems will be presented in depth in future work.

6. Conclusions and further work

This work presents the results obtained after carrying out a systematic literature review of literature whose aim was to identify and evaluate the current approaches for model consistency management. To do this, a total of 907 papers published in literature and extracted from the most relevant scientific sources were considered, of which 44 were eventually analyzed in depth, in accordance with the SLR process adopted.

The results of this SLR show that UML model consistency is a highly active and promising line of research, in which a great deal of quality work has been done, but in which some important gaps and limitations still exist, which should be tackled in future work. With regard to the new UML model consistency management proposals, we have three main recommendations to make (see Section 4 for more details). From our point of view, these proposals should:

(RQ1) Include a mechanism to extend the proposal in order to facilitate the managing of new models and inconsistency problems.
(RQ2) Tackle inconsistency problems related to vertical consistency, since they have been less frequently studied.
(RQ3) Fully integrate of all the features of the proposal within a CASE tool. This will thus permit its use and validation outside the academic scope.

Following these conclusions, we have presented a preliminary formal approach for the handling of inconsistency problems. This approach is based on transformation languages and rewriting logic and attempts to solve the limitations identified by the SLR while simultaneously including the strengths of the existing proposals. The main features included in this proposal are: extensibility, alignment with the MDA proposal, a formal basis and CASE tool integration.

The approach presented aims to fulfil the research challenges identified in the SLR through the definition of an extension mechanism by using the QVT Relations syntax. This mechanism will allow us to tackle any type of diagram, since the QVT Relations language can be applied over any metamodel (or DSL), in order to add new consistency checks and to define new ways of handling an inconsistency problems.

Furthermore, vertical inconsistency problems, which have not been studied in depth by the current approaches with regard to the SLR results, can be tackled in a natural manner thanks to the alignment of our proposal with MDA and transformation languages, which tackle models at different levels of abstraction. Since model transformation languages are well-known in this scope, the approach presented is particularly useful for modellers who are familiar with model-driven engineering processes.

Moreover, the use of rewriting logic in this approach, which manages all the elements as mathematical entities, offers a powerful manner in which to verify type properties and the correctness of the models or to use the inference mechanisms of this formalism

without losing the legibility, practicality, and expressivity of other known languages such as QVT Relations.

As regards future work, we first intend to extend the IQVT-*Maude* language to avoid having to consider the implementation details mentioned. The distance between QVT Relations and the IQVT-*Maude* language will thus be smaller still. Moreover, this language is beginning to be used in more complex examples and in other kinds of inconsistency problems. What is more, since rewriting rules allow us to transform model elements, their use in offering formal automatic support for model transformations within the scope of QVT and their applications to the verification and validation of transformations will also be studied in future work.

References

[1] OMG, Object Management Group, Unified Modeling Language: Superstructure, Version 2.1.1, Retrieved from: <http://www.omg.org/uml>, 2007.

[2] J. Muskens, R. Bril, M. Chaudron, Generalizing consistency checking between software views, in: 5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05), 2005, pp. 169–180.

[3] Z. Huzar, L. Kuzniarz, G. Reggio, J.L. Sourrouille (Eds.), Proceedings of Workshop on Consistency Problems in UML-based Software Development II, 2003.

[4] D. Schmidt, Guest editor's introduction: model-driven engineering, IEEE Comput. 39 (2) (2006) 25–31.

[5] OMG, MDA Guide Version 1.0.1, <http://www.omg.org/mda>, 2001.

[6] J. Simmonds, C.M. Bastarrica, A tool for automatic UML model consistency checking, in: Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering, Long Beach, CA, USA, 2005.

[7] B. Kitchenham, Guidelines for performing systematic literature reviews in software engineering, EBSE Technical Report EBSE-2007-01, Software Engineering Group, School of Computer Science and Mathematics, Keele University, UK and Department of Computer Science, University of Durham, Durham, UK, 2007.

[8] H. Ehrig, B. Mahr, Fundamentals of Algebraic Specification: Equations and Initial Semantic, Springer-Verlag, 1985. ISBN: 3-540-13718-1.

[9] G. Spanoudakis, A. Zisman, Inconsistency management in software engineering: survey and open research issues, in: S.K. Chang (Ed.), Handbook of Software Engineering and Knowledge Engineering, vol. 1, World Scientific Publishing Co., 2001, pp. 329–380.

[10] F.J. Lucas, F. Molina, A. Toval, A systematic review of UML model consistency management, Technical Report: I+D+I-2008, Departamento de Informática y Sistemas, University of Murcia, (Also available from: <http://express.in.um.es/fjlucas/>), December 2008.

[11] Y. Shinkawa, Inter-Model Consistency in UML Based on CPN Formalism, XIII Asia Pacific Software Engineering Conference (APSEC'06), 2006.

[12] Z. Huzar, L. Kuzniarz, G. Reggio, J.L. Sourrouille, Consistency problems in UML-based software development, in: N. Jardim Nunes, B. Selic, A. Silva, A. Toval (Eds.), UML Modeling Languages and Applications, UML 2004 Satellite Activities Lisbon, October 11–15, LNCS, vol. 3297, Springer-Verlag, Portugal, 2004. Revised Selected Papers.

[13] G. Engels, J.M. Kuster, R. Heckel, L. Groenewegen, A methodology for specifying and analyzing consistency of object-oriented behavioral models, in: Proceedings of Eighth European Software Engineering Conference Held Jointly with Ninth ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE-2001), September 2001, ACM Press, Vienna, Austria, 12, 98, pp. 186–195.

[14] J. Biolchini, P. Mian, A. Natali, G. Travassos, Systematic review in software engineering, Technical Report RT-ES679/05, Universidade Federal do Rio de Janeiro, Program de Engenharia de Sistemas e Computação, 2005.

[15] M. Gagalla, Benefits and problems of formal methods, in: Reliable Software Technologies – Ada-Europe 2004, Proceedings of the 9th Ada-Europe International Conference on Reliable Software Technologies, Palma de Mallorca, Lecture Notes in Computer Science, Springer, Spain, 2004, pp. 1–15.

[16] R. Wagner, H. Giese, U. Nickel, A plug-in for flexible and incremental consistency management, in: Proceedings of the International Conference on the Unified Modeling Language 2003 (Workshop 7: Consistency Problems in UML-based Software Development), San Francisco, USA, Technical Report, Ilmenau Institute of Technology, San Francisco, 2003.

[17] M. Schrefl, M. Stumptner, Behavior-consistent specialization of object life cycles, ACM Trans. Softw. Eng. Methodol. 11 (1) (2002) 92–148.

[18] A. Egyed, Fixing inconsistencies in UML design models, in: 29th International Conference on Software Engineering (ICSE'07), 2007, pp. 292–301.

[19] P.C. Supra, H. Mahanty, Ensuring consistency in relational repository of UML models, in: 10th International Conference on Information Technology (ICIT 2007), 2007, pp. 217–222.

[20] A. Egyed, UML/Analyzer: a tool for the instant consistency checking of UML models, in: 29th International Conference on Software Engineering (ICSE'07)-00, 2007, pp. 793–796.

[21] R.F. Paige, P.J. Brooke, J.S. Ostroff, Metamodel-based model conformance and multiview consistency checking, ACM Trans. Softw. Eng. Methodol. 16 (3) (2007) 11.

[22] H. Mougouyres, G. Mazer, A UML model consistency verification approach based on meta-modeling formalization, in: SAC'06: Proceedings of the 2006 ACM Symposium on Applied Computing, ACM, USA, 2006, pp. 1804–1809.

[23] A. Egyed, Instant consistency checking for the UML, in: ICSE'06: Proceedings of the 28th International Conference on Software Engineering, ACM, New York, NY, USA, 2006, pp. 381–390.

[24] D. Chiorean, M. Pasca, A. Cârcu, C. Botiza, S. Moldovan, Ensuring UML models consistency using the OCL environment, Electr. Notes Theor. Comput. Sci. 102 (2004) 99–110.

[25] J.M. Kuster, Towards inconsistency handling of object-oriented behavioral models, Electr. Notes Theor. Comput. Sci. 109 (2004) 57–69.

[26] P. Amaya, C. Gonzalez, J.M. Murillo, Towards a subject-oriented model driven framework, Electr. Notes Theor. Comput. Sci. 163 (1) (2006) 31–44.

[27] N. Amalio, S. Stepney, F. Polack, Formal proof from UML models, in: Formal Methods and Software Engineering, Proceedings of the 6th International Conference on Formal Engineering Methods, ICFEM 2004, Lecture Notes in Computer Science, Springer, Seattle, WA, USA, 2004, pp. 418–433.

[28] K. Diethers, M. Huhn, Voodoo: verification of object-oriented designs using UPPAAL, in: Tools and Algorithms for the Construction and Analysis of Systems, Proceedings of the 10th International Conference, TACAS 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, 2004, pp. 139–143.

[29] A. Egyed, Consistent adaptation and evaluation of class diagrams during refinement, in: Fundamental Approaches to Software Engineering, Proceedings of the 7th International Conference, FASE 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004 Barcelona, LNCS, Springer, Spain, 2004, pp. 37–53.

[30] G. Engels, R. Heckel, J.M. Kuster, L. Groenewegen, Consistency-preserving model evolution through transformations, in: UML 2002 – The Unified Modeling Language, 5th International Conference, Dresden, Germany, Lecture Notes in Computer Science, Springer, 2002, pp. 212–226.

[31] G. Engels, J.M. Kuster, R. Heckel, L. Groenewegen, Towards consistency-preserving model evolution, in: IWPSE'02: Proceedings of the International Workshop on Principles of Software Evolution, ACM, New York, NY, USA, 2002, pp. 129–132.

[32] J.M. Kuster, G. Engels, Consistency management within model-based object-oriented development of components, in: Formal Methods for Components and Objects, Second International Symposium, FMCO 2003, Leiden, The Netherlands, November 4–7, Revised Lectures, Lecture Notes in Computer Science, Springer, 2003, pp. 157–176.

[33] G. Engels, J.M. Kuster, R. Heckel, L. Groenewegen, A methodology for specifying and analyzing consistency of object-oriented behavioral models, SIGSOFT Softw. Eng. Notes 26 (5) (2001) 186–195.

[34] R. Heckel, J.M. Kuster, Behavioral constraints for visual models, Electr. Notes Theor. Comput. Sci. 90 (3) (2001) 257–269.

[35] L. Tryz, L. Kaculski, Assurance of system consistency during independent creation of UML diagrams, in: International Conference on Dependability of Computer Systems (DepCoS-RELCOMEX 2007), IEEE Computer Society, Szklarska Poreba, Poland, 2007, pp. 91–98.

[36] B. Graaf, A. van Deursen, Model-driven consistency checking of behavioural specifications, in: Proceedings – Fourth International Workshop on Model-Based Methodologies for Pervasive and Embedded Software, MOMPES, 2007.

[37] J.H. Hausmann, R. Heckel, S. Sauer, Extended model relations with graphical consistency conditions, in: Proceedings UML 2002 Workshop on Consistency Problems in UML-based Software Development, Blekinge Institute of Technology, 2002, pp. 61–74.

[38] K.M. van Hee, N. Sidorova, L.J. Somers, M. Voorhoeve, Consistency in model integration, in: Business Process Management: Proceedings of the Second International Conference, BPM 2004, Potsdam, Germany, 2004, pp. 1–16.

[39] P. Inverardi, H. Muccini, P. Pelliccione, Automated check of architectural models consistency using SPIN, in: 16th IEEE International Conference on Automated Software Engineering (ASE 2001), Coronado Island, San Diego, CA, USA, 2001, pp. 346–349.

[40] D. Khaltar, C.M. Krishna, U. Shrotri, R. Venkatesh, Visual specification and analysis of use cases, in: SoftVis'05: Proceedings of the 2005 ACM Symposium on Software Visualization, ACM, New York, NY, USA, 2005, pp. 77–85.

[41] R. Laleau, F. Polack, Using formal meta-models to check consistency of functional views in information systems specification, Inf. Softw. Technol. 50 (7-8) (2008) 797–814.

[42] V.S.W. Lam, J.A. Padget, Consistency checking of sequence diagrams and statechart diagrams using the pi calculus, in: Integrated Formal Methods, Proceedings of the 5th International Conference, IFM 2005, Eindhoven, The Netherlands, 2005, pp. 347–368.

[43] F.J. Lucas, A. Toval, A precise approach for the analysis of the UML models consistency, in: BP-UML'05: 1st International Workshop on Best Practices of UML, in 24th International Conference on Conceptual Modeling (ER 2005), Klagenfurt (Austria), ISBN: 3-540-29389-7. LNCS 3770, Springer.

[44] T. Mens, R.V.D. Straeten, M. D'Hondt, Detecting and resolving model inconsistencies using transformation dependency analysis, in: Model Driven Engineering Languages and Systems, Proceedings of the 9th International Conference, MoDELS 2006, LNCS, Springer, Genova, Italy, 2006, pp. 200–214.

[45] D.D.O. Ossami, J. F. Jacquot, J. Souquières, Consistency in UML and B Multi-view specifications, in: Integrated Formal Methods, Proceedings of the 5th International Conference, IFM 2005, LNCS, Springer, Eindhoven, The Netherlands, 2005, pp. 386–409.

[46] R.F. Paige, L. Kurtinskaya, J.S. Ostroff, J. Lancaric, BON CASE: an extensible CASE tool for formal specification and reasoning, J. Object Technol. 1 (3) (2002) 77–96.
[47] R.F. Paige, D.S. Kolovos, F. Polack, Refinement via consistency checking in MDA, Electr. Notes Theor. Comput. Sci. 137 (2) (2005) 151–161.
[48] H. Rasch, H. Wehrheim, Checking consistency in UML diagrams: classes and state machines, in: Formal Methods for Open Object Based Distributed Systems, Lecture Notes in Computer Science, Springer, 2003.
[49] Y. Shinkawa, Inter model consistency in UML based on CPN formalism, in: XIII Asia Pacific Software Engineering Conference (APSEC'06), 2006.
[50] G. Spanoudakis, K. Kozis, F. Dragazi, Evidential diagnosis of inconsistencies in object oriented designs, Int. J. Softw. Eng. Knowl. Eng. 14 (2) (2004) 141–178.
[51] G. Spanoudakis, H. Kim, Diagnosis of the significance of inconsistencies in object oriented designs: a framework and its experimental evaluation, J. Syst. Softw. 64 (2002) 3–22.
[52] R.V.D. Straeten, V. Jonckers, T. Mens, A formal approach to model refactoring and model refinement, Softw. Syst. Model. 6 (2) (2007) 139–162.
[53] R.V.D. Straeten, M. D'Hondt, Model refactorings through rule-based inconsistency resolution, in: SAC'06: Proceedings of the 2006 ACM Symposium on Applied Computing, ACM, New York, NY, USA, 2006, pp. 1210–1217.
[54] R.V.D. Straeten, Inconsistency detection between UML models using racer and nRQL, in: ADL'04 Third International Workshop on Applications of Description Logics, 2004.
[55] R. Van Der Straeten, T. Mens, J. Simmonds, V. Jonckers, Using description logics to maintain consistency between UML models, in: P. Stevens, J. Whittle, G. Booch (Eds.), UML 2003 – The Unified Modeling Language, Lecture Notes in Computer Science, vol. 2863, Springer Verlag, 2003, pp. 326–340.
[56] H. Wang, T. Peng, J. Zhang, K. Zhang, Consistency check between behaviour models, in: ISCIT 2005 – 5th International Symposium on Communications and Information Technologies 2005, Proceedings II, 2005.
[57] S. Yao, S.M. Shatz, Consistency checking of UML dynamic models based on petri net techniques, in: 15th International Conference on Computing (CIC'06) 0, 2006, pp. 289–297.
[58] W.L. Yeung, Checking consistency between UML class and state models based on CSP and B, J. Universal Comput. Sci. J. UCS 10 (11) (2004) 1540–1559.
[59] X. Zhao, Q. Long, Z. Qiu, Model checking dynamic UML consistency, in: Formal Methods and Software Engineering, Proceedings of the 8th International Conference on Formal Engineering Methods, ICFEM 2006, Lecture Notes in Computer Science, Springer, Macau, China, 2006, pp. 440–459.
[60] EMF. Project, Eclipse Modeling Framework (EMF), <http://www.eclipse.org/modeling/emf/>, 2007.
[61] OMG, MOF MetaObject facility specification, Object Management Group., Retrieved from: <http://www.omg.org/docs/ptc/04-10-15.pdf>, 2004.
[62] B. Beckert, T. Haure, R. Hähnle, D.R. Smith, C. Green, S. Ranise, C. Tinelli, T. Ball, S.K. Rajamani, Intelligent systems and formal methods in software engineering, IEEE Intelligent Syst. 21 (6) (2006) 71–81.
[63] H. Habrias, M. Frappier, Software Specification Methods, ISTE, 2006.
[64] B. Dobing, J. Parsons, How UML is used, Commun. ACM 49 (5) (2006) 109–113.
[65] J. Warmer, A. Kleppe, The Object Constraint Language: Precise Modelling with UML, Addison Wesley, 1999.
[66] J.L. Sourrouille, G. Caplat, Constraint checking in UML modeling, SEKE (2002) 217–224.
[67] A.J. Smith, The task of the referee, IEEE Comput. 23 (4) (1990) 65–71.
[68] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, C. Talcott, Maude 2.3 Manual, <http://maude.cs.uiuc.edu/>, 2007.
[69] OMG, MOF QVT Final Adopted Specification, Object Management Group, Retrieved from: <http://www.omg.org/docs/ptc/07-07-07.pdf>, 2007.
[70] F. Jouault, F. Allilaire, J. Bézivin, I. Kurtev, P. Valduriez, ATL: a qvt-like transformation language, 21th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2006, Portland, Oregon, USA.
[71] J.S. Cuadrado, J.G. Molina, Building domain-specific languages for model-driven development, IEEE Softw. 24 (5) (2007) 48–55.
[72] A.G. Kleppe, J. Warmer, W. Bast, MDA Explained: The Model Driven Architecture: Practice and Promise, Addison Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.
[73] P. Selonen, K. Koskimies, M. Sakkinen, Transformation between UML diagrams, J. Database Manag. 14 (3) (2003) 37–55.
[74] Borland, Together: Visual Modeling for Software Architecture Design, <http://www.borland.com/together>, 2007.
[75] Rational, Rational Software Architect, <http://www-306.ibm.com/software/awdtools/architect/swarchitect/>, 2008.
[76] Eclipse, Generative Modeling Technologies (GMT): MOFScript, <http://www.eclipse.org/gmt/>, 2007.