Universidad Rey Juan Carlos

Escuela Técnica Superior de Ingeniería Informática

Departamento de Lenguajes y Sistemas Informáticos II

M2DAT: a Technical Solution for Model-Driven Development of Web Information Systems

Author: Juan M. Vara Mesa

Thesis Supervisor: Esperanza Marcos Martínez

Móstoles, September 2009

Universidad Rey Juan Carlos

La Dra. Dª Esperanza Marcos Martínez, Profesora Titular de Universidad Departamento de Lenguajes y Sistemas Informáticos II de la la Universidad Rey Juan Carlos de Madrid, directora de la Tesis Doctoral: “M2DAT: A TECHNICAL SOLUTION FOR MODEL-DRIVEN DEVELOPMENT OF WEB INFORMATION SYSTEMS” realizada por el doctorando D. Juan Manuel Vara,

HACE CONSTAR QUE:

esta tesis doctoral reúne los requisitos para su defensa y aprobación

En Madrid, a 2 de Septiembre de 2009

Fdo.: Esperanza Marcos Martínez

“In theory, there is no difference between theory and practice.
But in practice, there is”

(Jan L. A. van de Snepscheut/Yogi Berra)

Abstract

During the last 20 years, there has been a continuous tendency towards raising the level of abstraction at which software is designed and developed. This way, assembly languages gave way to structured programming that yielded to object-orientation and so on. The last step in this line has been the Model-Driven Engineering (MDE) paradigm,that promotes the use of models as primary actors in the software development.

The underlying idea is to capture the system requirements and specification in high-level abstraction models that are automatically refined into low-level abstraction models. The latter takes into account the details of the targetting platforms and could be shown as the plans for the working-code. Indeed, such models are directly serialized into the working-code that implements the system. This way, automation comes as the other key of MDE: there is a need of tools for defining models, connecting them by means of model transformations, serializing them into code, etc.

During the last years, the impact of the MDE paradigm has resulted in the advent of a number of methodological proposals for Model-Driven Software Development (MDSD). According to the MDE principles, the authors of such proposals have developed the corresponding tools that should provide with the technical support for them. However, the absence of standards and their closed nature have resulted in tools providing with ad-hoc solutions that do not make the most of IDM's advantages in the form of less costly, rapid software development.

In this context, this thesis addresses the specification of M2DAT (MIDAS MDA Tool), a framework for semi-automatic model-driven development of Web Information Systems. To that end, instead of developing the technical support for each task comprised in a MDSD proposal, M2DAT integrates the isolated functionality provided by a set of existing tools for MDE tasks that will be used as building blocks.

This way, as part of this thesis we will define a conceptual architecture for MDSD frameworks. It will be an extensible, modular and dynamic architecture that promotes the integration of new capabilities in the form of new modules or subsystems and supports introducing desing decisions to drive the embedded model transformations. As well, since the proposed environtment follows a modular architecture, the development process to follow in order to build and integrate new modules will be defined.

Likewise, a set of methodological and technological decisions will be reasoned and justified to map the conceptual architecture to a technical design.

Finally, in order to prove the feasibility of the proposal and to show that it can be used in practice and how it should be done, a reference implementation will be provided. In particular, one of the modules of M2DAT, that supports the model-driven development of modern Database schemas will be developed.

In summary, M2DAT aims at solving some drawbacks detected in existing tools for supporting MDSD methodologies, mainly due to their isolated and closed nature: in contrast with previous works in the field, M2DAT will be easily extensible to ease the task of responding to new advances in the field. Likewise, it will be highly interoperable to simplify the use of the functionality provided by any other tool with M2DAT’s models. Finally, special attention will be paid to the management of model transformations in M2DAT, since they are the cornerstone of any MDSD methodological proposal.

Resumen

Durante los últimos 20 años la tendencía a elevar el nivel de abstracción con el que el software se diseña y construye ha venido siendo una constante. Los lenguajes de ensamblador dieron paso a la programación estructurada, que a su vez cedió el protagonismo a la orientación a objetos y así sucesivamente. El útlimo paso en esta dirección ha sido la aparición de la Ingeniería Dirigida por Modelos (IDM), cuya principal característica es el papel principal que juegan los modelos en el desarrollo de software.

La idea subyacente es recoger los requisitos y la especificación del sistema en modelos con un alto nivel de abstracción, que son automáticamente transformados en modelos de bajo nivel. Estos modelos, que consideran ya los detalles tecnológicos de la plataforma final y pueden contemplarse como los planos del software, son directamente transformados en el código fuente que implementa el sistema. De este modo, la automatización es otra de las claves de la IDM: se necesitan herramientas para definir modelos, para conectarlos mediante transformaciones, para generar código a partir de ellos, etc.

Durante los últimos años, el impacto de la IDM ha resultado en la aparición de numerosas propuestas metodológicas para el Desarrollo de Software Dirigido por Modelos (DSDM). De acuerdo a los principios de la IDM, los autores de dichas metodologías han desarrollado las herramientas que debían proporcionar el correspondiente soporte tecnológico para sus propuestas. Sin embargo, la ausencia de estándares y su naturaleza cerrada ha resultado en herramientas aisladas que proporcionan soluciones demasiado específicas y que no aprovechan por completo las ventajas de la IDM.

En este contexto, la tesis doctoral que se presenta aborda la especificación de M2DAT (MIDAS *MDA Tool*), un entorno para soportar el desarrollo semi-automático y dirigido por modelos de Sistemas de Información Web (SIW). Para ello, en lugar de desarrollar el soporte para cada una de las tareas que implica cualquier proceso de DSDM, M2DAT integra las funcionalidades aisladas que propocionan un conjunto de herramientas ya existentes. Es decir, dichas herramientas se utilizan como unidad de construcción para obtener un entorno integrado que dé soporte al proceso de desarrollo completo.

Así, como parte de la tesis se definira una arquitectura conceptual para entornos de DSDM. Será una arquitectura extensible, modular y dinámica que favorecerá la inclusión de nuevas funcionalidades como nuevos módulos o

susbsistemas y que soportará la introducción de decisiones de diseño que guién la ejecución de las transformaciones de modelos soportadas por la herramienta. Igualmente, dado que la herramienta sigue una arquitectura modular, se define el proceso de desarrollo a seguir para la construcción e integración de nuevos módulos.

Así mismo, un conjunto de decisiones metodológicas y tecnológicas, debidamente razonadas y justificadas, servirán para trasladar la arquitectura conceptual propuesta a un diseño técnico.

Finalmente, con el objetivo de demostrar la viabilidad de la propuesta y mostrar que puede llevarse a la práctica y cómo debe hacerse, se proporcionará una implementación de referencia. En particular, se desarrollará uno de los módulos de M2DAT, que soportará el desarrollo dirigido por modelos de esquemas de BD modernas.

Con todo ello, el objetivo final de M2DAT es solventar y/o paliar algunos de los problemas detectados en las herramientas existentes para dar soporte a metodologías de DSDM. Así, frente a dichas herramientas, que son de naturaleza eminentemente aislada, M2DAT será fácilmente extensible para responder a la aparición de nuevos avances en el campo de la IDM; tendrá un alto nivel de interoperabilidad, lo que posibilitará utilizar la funcionalidad proporcionada por cualquier otra herramienta para trabajar con los modelos elaborados con M2DAT; y pondrá especial atención en la gestión de las transformaciones de modelos, dado que constituyen el núcleo de cualquier propuesta de DSDM.

Acknowledgements

First and foremost, I would like to thank to Esperanza, my thesis supervisor. She introduced me to the research world and gave me the opportunity to involve myself in it; she has showed me, with her support and ideas, the successful path to the finishing of this PhD thesis; she has motivated me when I was depressed and she has helped me to keep my feet in the ground when I was too enthusiastic. All in all, I guess she knows me even better than myself. I guess it is enough with saying that, after all these years, she has became my friend.

I would like to thank also the rest of members of the KYBELE Research Group. Sometimes it was some of you, during some other periods, it was some others of you. At the end, all of you have contributed decisively to the finishing of this thesis. Specially those three girls ;-) that made this possible.

Thanks also to those two guys behind the scenes in the last stages: Emanuel and Alex. Modelling, coding and transforming with you has been a pleasure. I will miss the never-ending discussions and debates around the best way to *eclipse* any light ☺.

During these years, the friends from the University have been another way of escape: Diego, Ramón et al. you know how it pains and you know what it costs. Thanks also for sharing being there.

I would like to remember the people from INRIA. Jean and his guys taught me a lot about Model-Driven Engineering and the like. In addition, Fred taught me some french and Marcos taught me that you can have real friends from abroad ☺.

Special thanks to my family: my mother, Amada, my brother, Berny, my grandpa, Luis and my grandma, Ana. We are alone but together. You are always there when the end seems too far away. You are the only thing in which I trust and the only one that I know will never fail. Vane, you are becoming another one …

Of course, I could not forget my friends: we do not discuss about computer science, software engineering and the like. You made me leave all the stuff behind as soon as we met in *El Desván* or wherever. And I thank you a lot for this. You are *grandes* guys.

All of you have been the witnessess of my complaints, my results and achievementes, my failures and my frustations during all these years. For that and for all the rest, thanks to all.

Finally, when the time to finish this thesis comes, I would like also to remember my father. You are not here, but I hope you are proud of me today. In the end, I guess that deserving your recognition has always been there, like a leif-motif to go forward.

…………..Y ahora en español …………

Primero y sobre todo, gracias a mi directora, Esperanza. Ella me introdujo en esto de la investigación y me dio la oportunidad de ganarme la vida con ello. Con sus ideas y su apoyo me ha mostrado el camino para llegar al final de esta tesis (y soy muy consciente de que en algunos momentos, esa tarea ha resultado realmente desesperante). Me ha echado un cable cuando lo veía todo negro y me ha hecho poner los pies en la tierra cuando se me iba la cabeza. A estas alturas, creo que me conoce casí mejor que yo mismo. Supongo que bastará con decir que, después de todos estos años, se ha convertido en mi amiga.

Gracias también al resto de Kybelitos. A veces habeis sido unos, en otros épocas habéis sido otros. Pero todos vosotros habéis contribuido a que esta tesis llegase a su fin. Especialmente las tres maravillosas señoritas (VBV) sin las que esto NO habría sido posible.

Por supuesto, gracias a los dos chicos que han estado tras las bambalinas en la etapa final y cuya aportación ha sido decisiva: Emanuel y Alex. Modelar, programar y transformar con vosotros ha sido un placer y un lujo. Y echaré de menos las eternas discusiones en torno a la mejor forma de *eclipsar* cualquier lucecilla ☺

A lo largo de estos años, los amigos de la Universidad también han sido buenos compañeros de fatigas y doctorandos sufridores: Diego, Ramón et al. sabeis lo que duele y lo que cuesta. Gracias también por estar ahí.

Quiero recordar también a la gente del INRIA en Nantes. Jean y sus chicos me enseñaron un montón de cosas sobre esto de la Ingeniería Dirigida por Modelos (que al final he llegado a creerme, al menos en parte ☺). Fred además me enseño un poquito de francés y Marcos me enseñó que puedes hacer amigos de verdad en cualquier parte del mundo.

Obviamente, los agradecimientos especiales van para mi familia. Mi madre Amada, mi hermano Berny, mi abuelo Luis y mi abuela Ana han estado ahí siempre. Sois a lo que me agarraba los días realmente j…, en los que todo parecía demasiado malo, demasiado duro y demasiado lejos. Pero lo hemos hecho (una vez más). Sois lo único que sé que no me va a fallar nunca. Y sólo espero no fallaros a vosotros. Vane, bienvenida al club ;-) …

Y no me olvido de mis amigos: nunca discutimos de ordenadores, de Ingniería del Software o de cosas por el estilo … y no sabeis cómo os lo agradezco. Dejar atrás todas esas *paranoias* en cuanto entraba por la puerta de *El Desván* o plantabamos el pie en la pista de futbol era una auténtica y necesaria liberación. Sois grandes chavales. Y me habeis aguantado como campeones que sois ☺

Todos y cada uno de vosotros habéis sido los testigos de mis subidones y mis bajones, de mis éxitos y mis fracasos, de mis frustaciones y mis quejas (y soy un quejica compulsivo …). Por todo eso, y por todo lo demás, GRACIAS a todos.

Por último y antes de acabar, quiero acordarme de mi padre. Aunque no estás hace tiempo y me fastidie reconocerlo, nunca dejaste de estar. Merecerme tu respeto y demostrarte que no era tan … fue siempre una de las principales razones para tirar pa'lante y echarle lo que había que echarle. Espero que hoy te puedas sentir un poquitín orgulloso del que suscribe …

Index

List of Figures

List of Tables

Introduction

In this thesis, a technical solution for model-driven development of Information Systems is proposed.

The first section of this chapter (section 1.1) introduces the motivations that led to the decision of undertaking this work as well as its main contributions. Section 1.2 states the main hypothesis and the objectives directly derived from it, where as section 1.3 describes the context in which this work has been developed, referring mainly to the research projects. Finally, section 1.4 summarizes the research method followed and section 1.5 provides with a general overview on the the rest of this dissertation.

1.1 Problem Statement and Approach

By the end of 2000, a new way of conceiving software development resulted in a mare magnum of acronyms (MDE, MDSD, MDD, DSL, MIC, etc.) that served to refer to a number of approaches sharing a common basis: to boost the role of models and modelling activities at the different steps of the development cycle. The main feature of the new paradigm, Model-Driven Engineering (MDE, [41]), was focusing on models rather than in computer programs. Indeed, MDE is a natural step in the historical tendency of software engineering towards raising the abstraction level at which software is designed and developed. Assembly languages gave way to structured programming languages that yielded to object-orientation and so on.

Notice that, though models have always been considered in software development, they have been traditionally used as simple documentation, and in the best case, they have served to generate a reduced skeleton of the final code (Rational Rose was the perfect example on this line [173]). From this point of view, models were discarded as soon as the corresponding development phase was finished, and they were not updated to reflect the changes made in subsequent models or in the working code.

The landscape has changed drastically with the advent of MDE. MDE practitioners shift their focus from coding to modelling. There is a swing towards defining accurate models that capture all the requirements and specifications about the system to build as well as the platform where it will be deployed. To that end, high-level models are subsequently refined into low-level models, until their level of detail is that of the underlying platform. Finally, the working-code for the whole system (and not only a skeleton) is automatically generated from those models.

However, the only way to get a full return of MDE promises of faster, less costly software development, was automating any model-driven software development proposal. This way, automation came as one of the keys of MDE [21, 134]. As a result, a number of **tools for supporting MDE tasks** have arisen during the last years to automate each task related with MDE. Consequently, since MDE is based on the use of models, one can find tools to define and use new modelling languages. Because model transformations are the key to bridge those models, there exist a number of tools or languages for model transformation. Given that models consistency is essential since now they are the driving force in the development process, several tools for model checking have appeared, etc. Notice that each one of these tools aims at providing with generic support for one concrete task among all the different tasks related with a model-driven development process, i.e. they focus on a subset of the functionality needed to implement a model-driven development proposal. For instance, the most commonly adopted model transformation language, ATL [184], fall in this category of tools for MDE tasks.

On the other hand, the impact of MDE has given rise to a number of Model-Driven Software Development (MDSD) methodologies. These methodologies are based on the definition and use of different modelling languages (whether general or special-purpose) to model and capture different parts of the system at different levels of abstraction. As a response, a new group of tools appeared to support those methodologies: the **tools for supporting MDSD methodologies** are software development environments that provide with toolkits to work with the specific set of interrelated models defined in the corresponding methodology in order to generate the working-code of a software system. One of the most recognised tools falling in this category is ArgoUWE [195], a CASE tool that aimed at implementing the UWE methodology [198].

The efforts that the developers behind those initial MDSD methodologies have dedicated to build the technical support to automate them have resulted in a number of isolated tools that provide with ad-hoc solutions. Their closed nature and the absence of standards when they started to be developed prevented them from taking advantage from the advances in the field and the capabilities provided by tools for MDE tasks. For instance, in the absence of model transformation engines when they were developed, they use to hard-code model transformations or constraints checking.

As a result, there is a need for building new tools to support MDSD methodologies that integrate the functionality provided by the existing tools for MDE tasks. In other words, tools for MDE tasks have to be used as building blocks in order to develop an integrated environtment that implements a MDSD methodology.

How the building of such environment is to be addressed? First, defining a conceptual architecture that abstracts from technical underpinning; next, mapping the conceptual architecture to a technical design and finally, providing with a reference implementation of such technical design.

In this sense, it is worth mentioning that nowadays, the strength of the MDE paradigm has resulted in a trend towards building this kind of integrated environtments to support MDSD methodologies. However, when we first addressed the development of this thesis there were no such type of tools. As it will be shown in Chapter 2, the trend is quite recent. Indeed, existing tools for supporting MDSD methodologies that were developed to run in an isolated way are moving to turn theirselves into integrated and extensible tools like the one that will be presented in this dissertation.

Besides, the innovative nature of MDE oblies to put special attention on a set of traditional requirements related with the development of software engineering tools. Under the light of MDE, extensibility, interoperability and customization become even more relevant when building the support for a MDSD methodology.

In fact, a tool supporting a MDSD methodology shall be rather **extensible** in order to response to the advent of new advances in the field. For instance, both definitions of dynamic semantics and formal specifications are gaining acceptance as a way towards model execution and simulation [319]. Technical support for these tasks is still rather immature. However, when it is mature enough, a tool supporting a MDSD methodology should be ready to be extended in order to support formal specifications of models and attachment of semantics definition.

Besides, if a desired functionality is already implemented in some other tool, it might be preferable integrating its use in the new tool that will support the methodology, instead of hard-coding directly such functionality. To that end, the new tool has to be able of handling models created with the tool supporting the methodology. Therefore, **interoperability** becomes a crucial feature to be supported by tools supporting MDSD methodologies.

As well, although the objective is at automating the whole development process proposed in the methodology, a customizable process that gives the designer the option of introducing design decisions to drive the development process at any stage, is also recommended [246]. Hence, we need a way of introducing support for **design decisions** without lessening the level of automation. Apart from the design decisions spread over the models handled along the process, the only way of introducing design decisions is supporting model-to-model and model-to-text customizable transformations.

In this context, the thesis presented addresses *the specification of a framework for semi-automatic model-driven development of web Information Systems*. To that end, this thesis will introduce *M2DAT (MIDAS MDA Tool)*, a tool for MDSD based on the methodological proposals of MIDAS, a model-driven methodology for *Web Information Systems (WIS)* development.

As part of this thesis, we will define a conceptual architecture for MDSD frameworks. It will be an extensible, modular and dynamic architecture that promotes the integration of new capabilities in the form of new modules or subsystems and supports introducing desing decisions to drive the embedded model transformations. As well, a systematic approach to build those modules will be defined.

Likewise, the conceptual architecture will be mapped to a technical design. This task implies a set of decisions about which is the best technology for each task and how it should be used. This way, a set of technological decisions will be made, like which is the most suitable tool for implementing the new modelling languages; the most convenient approach to develop the transformations that have to bridge those languages; the most convenient model transformation engine among those following the selected approach, etc. To that end, a review of existing technology will be made according to a set of criteria defined to fulfil the requirements of the planned framework (extensibility, interoperability, customizable transformations, etc.). As a result, a selection of technology will be obtained. It will identify the component that will provide with support for each specific task and the design decisions that will drive the mapping between conceptual and technical design.

Finally, a reference implementation will be provided to prove the feasibility of the proposal and show that it can be used in practice [95]. In particular, one of the modules of M2DAT will be developed. The module for model-driven development of modern database schemas, M2DAT-DB (MIDAS MDA Tool for DataBases) will support the development of Object-Relational and XML Schemas DBs. The construction of M2DAT-DB will serve to show that both, the conceptual and technical design of M2DAT, as well as the design decisions and the development process defined, are suitable for implementing MDSD proposals. In fact, M2DAT-DB bundles a wide range of model transformation types (vertical, horizontal, PIM2PSM, PSM2PSM, PSM2CODE, etc.) as well as the rest of tasks that need to be automated when implementing a MDSD methodology (definition of abstract and concrete syntax for DSLs, graphical editors, models validators, etc.)

1.2 Hypothesis and Objectives

In the following, the main hypotheses in this thesis as well as the objectives derived from it are put forward.

The **hypothesis** formulated in this dissertation is that "*it is feasible to provide with a technical solution for the construction of a framework supporting semi-automatic model-driven development of Web Information systems using existing tools and components in the context of MDE*"

Hence, the **main objective** of this thesis, directly derived from the hypothesis, is: "*to provide with a technical solution to build a framework for semi-automatic model-driven development of Web Information Systems using existing tools and components in the context of MDE*"

This objective is broke down into a set of partial objectives:

O1. Analysis and evaluation of existing technologies (tools for MDE tasks) in order to identify the most suitable to build a framework for model-driven development of WIS. According to the specific tasks that building such a farmework entails, we can split this objective as follows:

- **O1.1.** Analysis and evaluation of (meta)modelling tools.
- **O1.2.** Analysis and evaluation of existing model-to-model transformation engines, stressing support for introducing design decisions in the mapping process.
- **O1.3.** Analysis and evaluation of model-to-text transformation engines (also known as code generators).
- **O1.4.** Analysis and evaluation of tools supporting the rest of specific tasks in MDE contexts, such as graphical/textual editors' development or constraints checkers for models.

O2. Analysis and evaluation of existing frameworks supporting model-driven software development.

- **O2.1.** Analysis and evaluation of existing frameworks for model-driven development of Web Information Systems.
- **O2.2.** Analysis and evaluation of existing frameworks for model-driven development of modern database schemas (object-relational and XML).

O3. Specification of the conceptual architecture of M2DAT framework.

O4. Selection of the technologies to be used for M2DAT.

O5. Specification of the technical design of M2DAT.

O6. Specification of the development process for each M2DAT module.

O7. Validation of the technical design of M2DAT. To that end, two main sub-objectives are identified:

- **O7.1.** Construction of one of M2DAT's modules to achieve a **proof of concept** for the proposal (conceptual architecture and technical design).
- **O7.2.** Development of a set of case studies using M2DAT-DB.

1.3 Research Context

All the works of the research group in which the Ph.D. candidate is integrated, as well as this thesis itself, spin around a common objective: the specification of MIDAS [1, 104, 105, 121, 211, 355, 366], that provides with an architecture-centric methodological framework (*ACMDA* in fact) for model-driven development of Web Information Systems, following a Service Oriented approach. The current version of MIDAS architecture is shown in Figure 1-1

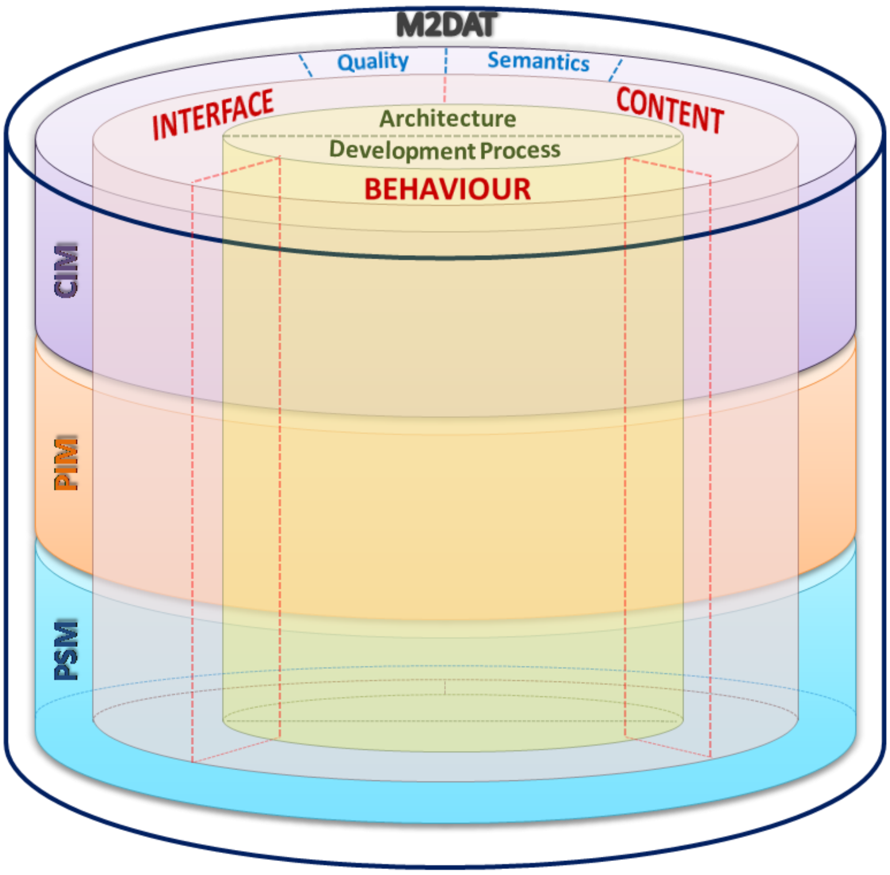


Figure 1-1. MIDAS Architecture overview

This models architecture is based on the MDA principles [264] and is defined upon a multidimensional basis that spreads through several abstraction levels and concerns of the system development. For each dimension or development concern, the models are considered together with the transformation rules between models and the influence of each model on the rest. As mentioned, MIDAS architecture can be considered from different dimensions:

- **Vertical dimension**. This one comes directly from the proposal of MDA defining three abstraction levels: Computation Independent Models (CIM), Platform Independent Models (PIM) and Platform Specific Models (PSM). This way, MIDAS moves down from the concepts associated to the problem domain gathered in the CIM models to the system representation according to specific features of the targeted platform by means of PSM models.

- **Core of the models architecture**. Since the architecture plays a guiding role, its models make up the core of the development process. Indeed, the architecture specifies features affecting not only one aspect of the system but all of them.
- **Inner concentric layer**. The models in this layer are organized according to the main concerns traditionally involved in the development of any Information System. This comprises the modelling of the Content, the Behaviour and the Interface, such as the hypertext modelling in Web applications.
- **External layers**. Advances in information technology have given rise to considering new aspects when developing information systems, such as the Semantics, related with the use of ontologies. These are included in MIDAS as orthogonal aspects for which a new set of models is defined.
- **Tool – M2DAT**. Finally, in order to implement the MIDAS methodological proposal, an extensible and interoperable MDSD framework, so-called M2DAT (MIDAS MDA Tool) has to be developed, providing support for each model comprised in the models architecture of MIDAS.

In the scope of MIDAS four Ph.D. theses have been previously developed. The most recents being the ones from Dr. César Acuña, which focuses on the semantics concern of MIDAS and the one from Dr. Valeria De Castro that focuses on the behaviour concern. Besides, apart form the present thesis, there are three more in progress. In particular, the thesis of Marcos Lopez is on its final stage. It tackles the development of Software Architectures from a Service-Oriented perspective and using a Model-Driven approach. The thesis of Veronica Bollati focuses on the definition of a common metamodel for model transformation languages. Finally, Elisa Herrmann's thesis is focused on the improvement of code generation mechanisms in the framework of MIDAS

On its turn, the last point from the description of MIDAS architecture will be tackled in this thesis. To that end, the present thesis will address the definition and design of the technical solution to automate MIDAS methodology.

Notice that the development of the technological support for MIDAS will make the most of the aspects separation that offers MIDAS architecture. Indeed, MIDAS architecture can be shown as a methodological approach composed of small methods offering solutions for specific problems or tasks, such as hypertext or database development. To that end, each method proposes a set of interrelated models. Those methods are combined and integrated to give rise to the final result: the information system.

Accordingly, M2DAT development will be addressed as a set of isolated subsystems providing with similar functionalities for a set of interrelated models. Such subsystems or modules will be later integrated by means of model transformation and model weaving techniques. Besides, MIDAS' modular architecture is devised to promote extensibility of the framework by inclusion of new concerns. Accordingly, the supporting framework, M2DAT, has to be also open to integrate support for the new concerns. Hence, extensibility is a must for M2DAT.

Furthermore, as Figure 1-1 shows, the M2DAT framework will act as the binding force that connects the different methods that compose MIDAS. Hence, interoperability is also mandatory for M2DAT.

In the end, the present thesis will serve as building basis for other research works focused on covering other aspects of MIDAS architecture. To that end, this thesis will perform a study of existing technology to choose the most convenient to address each task. As well, the way they have to be integrated to conform a technical solution to implement any new method incorporated in MIDAS will be specified. For instance, people working on the semantics concern will use the technical solution specified in this thesis to develop a set of DSLs for modelling the semantics of the information system. To that purpose, they will follow the techniques described here and will use the technical components appointed for each task. Besides, because of its interoperable nature, M2DAT will provide with immediate integration of the models comprised in the new method with those from the already implemented methods.

1.3.1 Research Projects and Stages

Throughout the development of this thesis, the Ph.D. candidate has done two research stages (see Figure 1-2). During the first one, he spent three months in the ALARCOS group from the University of Castilla-La Mancha (UCLM), working under the supervision of Dr. Francisco Ruiz in the field of CASE tools for MDA. The second one was a four-months stage in Nantes, where integrated in the ATLAS group, led by Professor Jean Bézivin, the Ph.D. candidate worked mainly on the studying ot the ATL language, the use of weaving models to introduce design decisions in model transformations and the automation of model migration by means of model-driven tecniques.

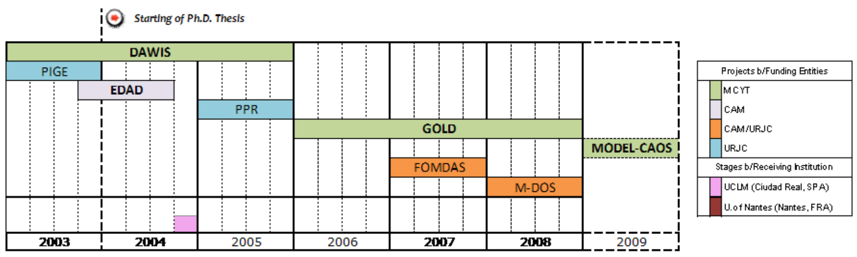


Figure 1-2. Ph.D. Thesis Research Context

Besides, along this period, the Ph.D. candidate, integrated in the Kybele research group from the Rey Juan Carlos University, has worked in a series of research projects. In particular, the work undertaken for this dissertation has been framed mainly in three interrelated projects: DAWIS, EDAD and GOLD.

DAWIS [TIC 2002-04050-C02-01], funded by Ministry of Science and Technology, was a coordinated project joint with the Technical University of Madrid between 2002 and 2005. It was focused on the systematic and semi-automatic development of Web portals providing integrated access to multiple digital libraries. The tasks of the Ph.D. candidate lied on the construction of bridges between different storage formats and the use of XML DBs for file management.

The proposals of DAWIS were implemented in, **EDAD** [07T/0056/2003 1] a project co-financed by the Regional Government of Madrid and the European Community that evolved during the 2003 and 2004. In the context of this project, the Ph.D. candidate worked on the construction of model-driven tools for Web Services and XML Schema development.

Finally, the **GOLD** project [TIN2005-00010], the main frame of this thesis, was also financed by the Ministry of Science and Technology and took place from the beginning of 2005 to the end of 2008. The main objective of this project was the construction of a platform for Web Information Systems development and its application to a system for medical images management. The Ph.D. candidate has been responsible for providing with a technical solution extensible and interoperable to build such platform. To that purpose, he has been involved in the application of model-driven solutions to all the aspects comprised in the design and construction of this platform, especially on those related with the development of model transformations. Hence, the results of the GOLD project comes mainly from the work undertaken in this thesis.

The continuation of GOLD is the **MODEL-KAOS** project, funded by the Ministry of Science and Technology [TIN2005-00010]. It aims at adapting the proposals of GOLD accordingly to the Service-Oriented paradigm [285].

As well, the Ph.D. candidate has been involved in the **FOMDAS** [URJC-CM-2006-CET-0387] and **M-DOS** [URJC-CM-2007-CET-1607] projects, co-funded by the Rey Juan Carlos University and the Regional Government of Madrid. The former was focused on formalizing part of the metamodels and model transformations specified in GOLD whereas the latter addressed the adaption of GOLD proposals to the Service Oriented paradigm.

1.4 Research Method

The different nature of engineering disciplines from that of empiric and formal disciplines does not allow the direct application of classical research methods to software engineering research.

The research method followed in this thesis is adapted from the one proposed in [223] for research in Software Engineering. It is based on the hypothetical-deductive method of Bunge [67] that is composed of several steps that, due to its genericity, apply to any kind of research.

As Figure 1-3 shows, the definition of the research method is a step in the method itself. It is needed since each research process has its own features. Hence there is no universal method that apply to any research work.

The most important phase of this method is the relosution and validation phase. Hence, next section provides with a wide overview on this matter.

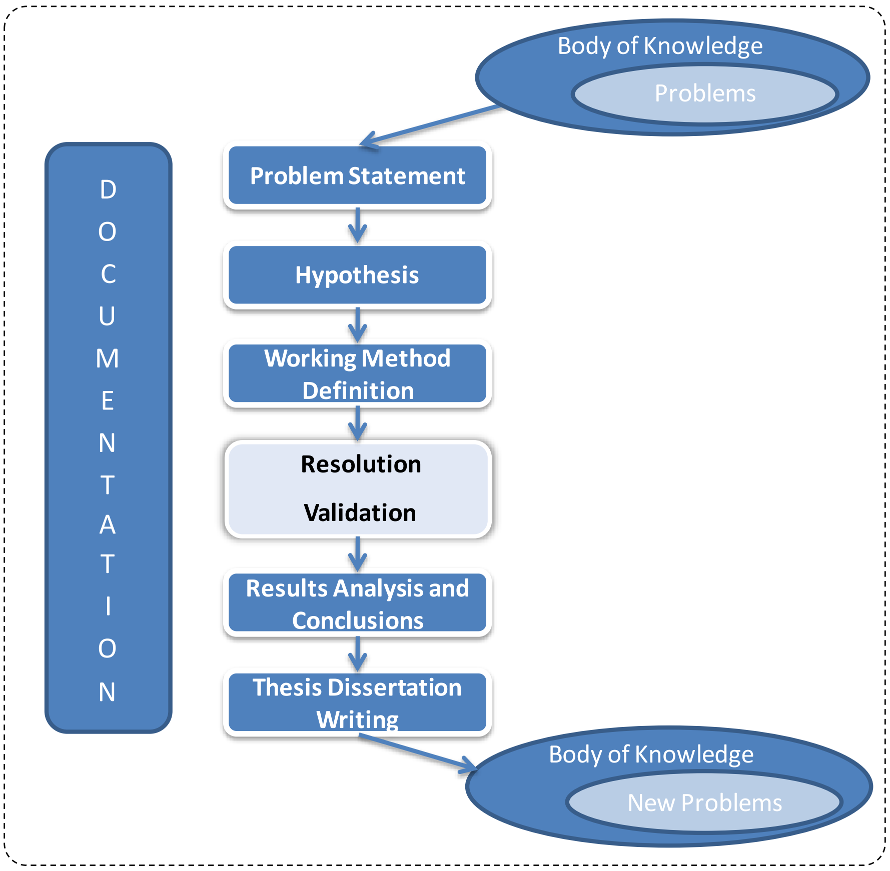


Figure 1-3. Research Method

1.4.1 Resolution and Validation Method

The resolution and validation method followed in this thesis is somehow adapted from the traditional waterfall [300], the Rational Unified Process [177] and the. Figure 1-4 shows a simplified overview of the method.

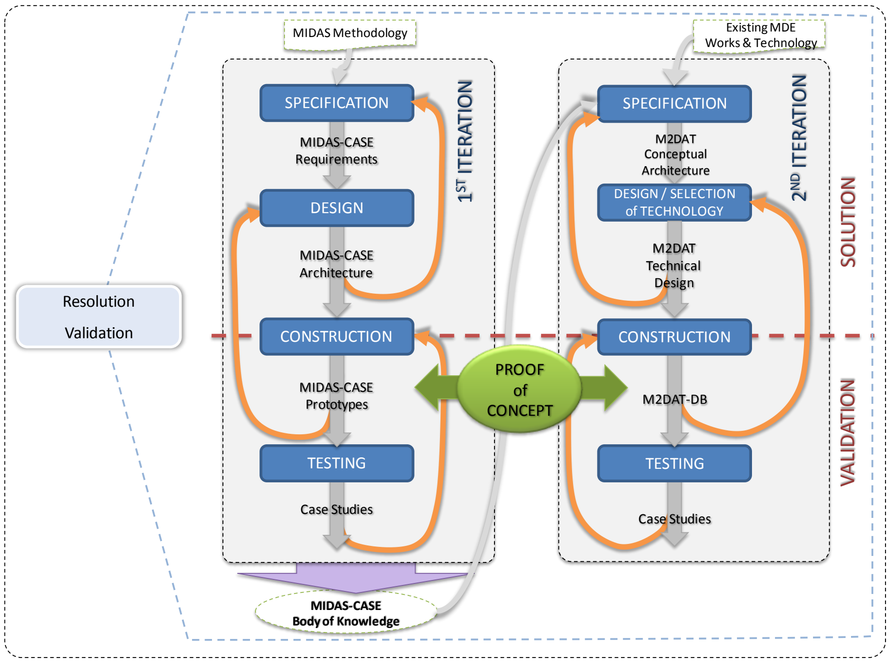


Figure 1-4. Resolution and Validation phase on the research method

In essence, two big iterations can be identified. In turn, these were composed of several iterations each one. A brief description of the work carried out in each iteration follows.

1.4.2 *First Iteration: MIDAS-CASE development*

During the **specification** phase of the first iteration, the existing works spinning around CASE tools were reviewed along with the MIDAS methodology. The aim was at identifying, on the one hand, the needs related with supporting MIDAS and on the other hand, to identify if existing tools could fit those needs. Such review resulted in the decision of building a new framework to support the graphical representation of all the models comprised in MIDAS, the automatic mapping between them and the automatic code generation from those models. Moreover, the use of an XML DB repository for model management was planned. In addition, two desired features were identified in order to support MIDAS open nature: the new framework had to be easily extensible and modular .

The **design** phase was mainly related with defining the architecture of the framework according to the requirements stated during the specification phase. Besides, the technical components to be used were identified and the development process to follow in order to build each module was defined. The main output of this phase was MIDAS-CASE architecture. It merged the conceptual description with the technical decisions.

To validate the result of the design phase, two prototypes were built during the **construction** phase: MIDAS-CASE4WS and MIDAS-CASE4XS supported the modelling of Web Services, respectively XML Schemas, with extended UML and the serialization of models into working-code (WSDL and XSD). They constitute the **proof of concept** for MIDAS-CASE architecture.

Finally, the **testing** phase consisted in the development of several case studies with the prototypes built. Such case studies served to assess on the feasibility and utility of the proposal in order to improve both the architecture and the development process for building MIDAS-CASE modules.

Note that each step on the process provides with continuous feedback over the previous ones. For instance, the findings gathered during the construction of MIDAS-CASE prototypes influenced the design phase in order to refine MIDAS-CASE architecture.

1.4.3 Second Iteration: M2DAT development

After finishing MIDAS-CASE development a new iteration (in turn comprised of a number of inner iterations) was undertaken. The objective was to incorporate and consider advances in the field of MDE and take the most from the lessons learned during the first iteration in order to solve the main drawbacks detected in tools supporting methodological proposals for MDSD.

Hence, the **specification** phase made an exhaustive review on existing support for MDE tasks ((meta)-modelling tools, model transformation engines, etc.) as well as the lessons learned from MIDAS-CASE project. A relevant conclusion was the convenience of separating the definition of the architecture of the framework at a higher abstraction level from its technical description. Thereby, the main output of this phase was the definition of the conceptual architecture of M2DAT, the new version of the tool to support MIDAS methodology, and the technical knowledge needed to address the design phase.

Lately, in the **design** phase the conceptual architecture was refined to a technical design according to the knowledge collected during the reviews from the specification phase. In contrast with MIDAS-CASE architecture, M2DAT was built by integrating a number of exiting tools for MDE tasks on top of the Eclipse Modelling Framework. Working this way, the result is a highly extensible framework, capable of integrating new capabilities as long as they are delivered. Likewise, the development process proposed to address the development of new modules according to the technical design of M2DAT was defined.

Such design guided the construction of the **proof of concept**, M2DAT-DB, during the **implementation** phase. It served as reference implementation to prove and validate the technical design of M2DAT.

Finally, a battery of case studies developed with M2DAT-DB during the **testing** phase helped on the refinement of the proposal. For instance, the need for parametrizable model transformations was detected during the development of those case studies. Consequently, the technical design of M2DAT and the development process for M2DAT's model transformations were modified according to the emerging need.

1.5 Thesis Outline

Remaining chapters of this thesis are organized as follows:

- **Chapter 2** provides with a complete overview on the state of the art. To that end, section 2.1 introduces the previous concepts related with this work. Next, three big groups of previous works are reviewed. Existing tools for MDE tasks are reviewed in section 2.2. A detailed review on model transformation engines is performed in section 2.3 while section 2.4 reviews existing frameworks for MDSD of Web Information Systems and modern database schemas.
- **Chapter 3** is focused on presenting the result of the first iteration of the research method followed, MIDAS-CASE (see section 1.4). To that purpose, section 3.2 describes MIDAS-CASE architecture and the process proposed to develop MIDAS-CASE modules. Section 3.3 introduces MIDAS-CASE prototypes: MIDAS-CASE4WS and MIDAS-CASE4XS. Besides, two case studies using each module are presented. Finally, section 3.4 puts forward the main conclusions and lessons learned gathered from developing MIDAS-CASE.
- **Chapter 4** presents the technological proposal acting as the basis of this thesis: M2DAT conceptual architecture, design decisions to map it to a technical design and development process for new modules. To that purpose, section 4.1 brings forward the conceptual architecture of M2DAT, adapted from that of MIDAS-CASE. In the remaining of the chapter, it is described how the conceptual architecture is refined into a technical design by selecting

the approach and technology for each specific task. The result is a framework that integrates a number of existing tools for specific MDE tasks using Eclipse and the Eclipse Modelling Framework as a meeting ground. To that end, each section provides with a discussion on the different methodlogical and technical options to address the common tasks related with deploying model-driven methodologies and reasons out the ones selected to build M2DAT.

- **Chapter 5** addresses the validation of M2DAT specification using M2DAT-DB, the reference implementation for M2DAT, that serves as proof of concept. It serves to introduce the way each specific task is to be implemented in M2DAT according to the methodlogical and technological decisions made on the previous chapter. Thereby, after providing an overview of M2DAT-DB functionality, this chapter deals with the way new DSLs are built on M2DAT (both abstract and concrete syntax). Next, a set of common scenarios that arose when developing model transformations and the way they are addressed in the context of M2DAT is described. Finally, ways of using M2DAT capabilities in a user-friendly manner by extending the Eclipse GUI are presented.
- Finally, **Chapter 6** concludes by summarizing the main contributions of this thesis. To that purpose, it provides an analysis of the results and reviews the publications that serve to contrast them both on national and international forums. Besides, it raises a number of questions for future research and puts forward the directions to follow for further work.
- In addition, Appendix A provides with a summary of this dissertation in Spanish; Appendixes B and C provide with more detailed discussions on graph-based and model-to-text transformations. Appendix D presents the main Case Study used along this dissertation to show M2DAT-DB’s capabilities. Finally, Appendix E collects the bibliographical and electronic references used along this dissertation and Appendix F summarizes the acronyms spread over the text.

State of the Art

During the last years, the popularity of Model-Driven Engineering (MDE) has given rise to the advent of many tools and frameworks providing with all or a part of the functionality needed to deploy MDE proposals. This chapter aims at providing an overview on them.

In order to make a comparative, the artefacts to compare should own the same nature. Therefore, we have splitted the state of the art on the following categories.

- **Tools supporting MDE tasks**. Here we give a brief overview on the main tools or projects focused on providing support to automate each task related with MDE. Thus, this section covers different types of tools, like metamodelling environments or frameworks for development of graphical editors. Due to its special relevance, we include a dedicated section for the Eclipse Modelling Framework (EMF, [66, 161]).
- **Model Transformation languages.** Since model, transformation is the cornerstone of MDE [52, 318], we provide with a wide study on existing model transformation languages. Both model-to-model and model-to-text transformation languages fall in this category.
- **Tools for Model-Driven Development of software.** This category groups together tools supporting model-driven software development for specific domains. Given the nature of this thesis, we will focus just on tools supporting methodological proposals for Web Information Systems (WIS) development and tools supporting model-driven development of (modern, OR and XML) database schemas.

The conclusions obtained from the state of the art are very relevant since this thesis is focused on designing and building and MDE framework. In fact, one of the main tasks to design M2DAT was selecting the right technology. Thus, the reasoning spread over the following sections will be revisited all along this document, especially in Chapter 4, where M2DAT technical design is presented.

Finally, before diving into review of exiting works, we would like to define a set of terminology to use along this dissertation. Please, note that we do not mean that this is the only one, but one consistent and valid for our purposes.

2.1 Previous Concepts

From the very first days of software engineering, researchers have had problems to make an agreement on a common terminology [63]. The problem gets worse in front of a fledging topic like Model-Driven Engineering, that is still far from being an established engineering discipline [319, 346]. Therefore, the following sections give a set of definitions and overviews for some common terms related whith Model-Driven Engineering. This is a needed step, since the success of the proposal has given rise to many buzzwords like metamodelling, domain specific modelling, concrete or abstract syntax and the like. We need a common understanding on those terms before diving into the study of existing works in the field.

2.1.1 MD Acronyms*

Applying Model-Driven techniques to the task of software development is becoming a hot topic today. As a result, a lot of initiatives and proposals have emerged on this field during the last years. However, the advent of so many proposals turned out in a mare magnum of acronyms to refer to the same approach. Different authors use different names to refer to (almost) the same thing. Thus, we may talk about Model-Driven Software Development [328] (MDSD), Model-Driven Development [166] (MDD), Model-Driven Engineering [41] (MDE), Model-Based Software Engineering [319] (MBSE) and so on. Along this dissertation we will use the term MDE as a common moniker for all of them, following the idea gathered in [110].

2.1.2 On CASE Tools, Frameworks and Components

Following Ambler's recommendation [15], throughout this dissertation we will use the term CASE (computer aided system/software engineering) tool to refer to software-based modelling tools.

In addition, we will refer indistinctively to frameworks and tools and will use the term component to refer just to a software module that encapsulates some specific functionality integrated into some framework. For instance, the Eclipse Modelling Framework (EMF) might be referred as a component of the Eclipse framework. As well, each EMF subproject, like the Validation Framework would be a component of EMF, and so on.

2.1.3 Models and Metamodels

There is a vast amount of definitions of **what a model is** (see [209] for a complete review on them). However, one can extract some common features to all of them that allow providing with a *standard* definition: a model is always an abstraction of something that exists in reality, i.e. details are left out, and it can be used to produce the reality modelled.

Following this line, a common issue to traditional engineering disciplines has been the definition of models as a previous step to the construction of the system. Such models act as the plans for the system under development and provide a specification that allows describing its structure and behaviour. Likewise, modelling has been widely adopted as a common practice in Software Engineering. Software models serve to visualize how the software system should look, specify its structure and its behaviour, guide its implementation and document the design decisions that drive the development process. In this sense, Kleppe et al. [193] provides with a commonly accepted definition of **software model**: "A model is a description of (part of) a system written in a well-defined language. A well-defined language is a language with well-defined form (syntax), and meaning (semantics), which is suitable for automated interpretation by a computer".

As it happens with the plan of a building, the model of a software system has to be precise enough to avoid errors when moving from the specification, i.e. the plan or the model, to the real system, i.e. the building or the software system. Thereby, strong efforts have been put to provide with modelling languages and notations that allows the definition of precise models. A key part on the rigorous definition of models is the specification of which are the allowed modelling elements and how they can be combined to create a new model. This knowledge is collected in a metamodel.

A **metamodel** is the model of a modelling language [315]. That is, a metamodel makes statements about what can be expressed in the valid models of a certain modelling language. Since a metamodel is nothing but another model, it might be expressed using the same modelling language that it defines. In that case, expressions in the metamodel are represented in the same language that describes the metamodel. This metamodel is called reflexive metamodel or **metametamodel**. Figure 2-1 depicts the conformance relationships between the different types of models. Such conformance relationship means that a model is defined according to the rules collected in another model.

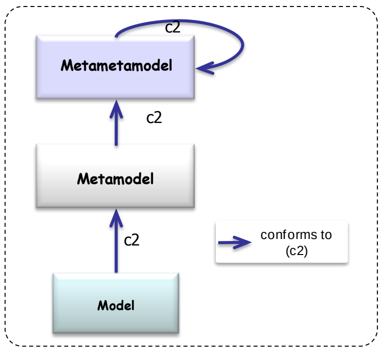


Figure 2-1. Modelling and Metamodelling

This way, any **terminal model** conforms to another model known as metamodel that, in turn, conforms to another model. In addition, the latter conforms to itself, thus it is called a metametamodel.

For instance, Figure 2-2 shows a simplified metamodel for modelling relational database schemas and a conforming model.

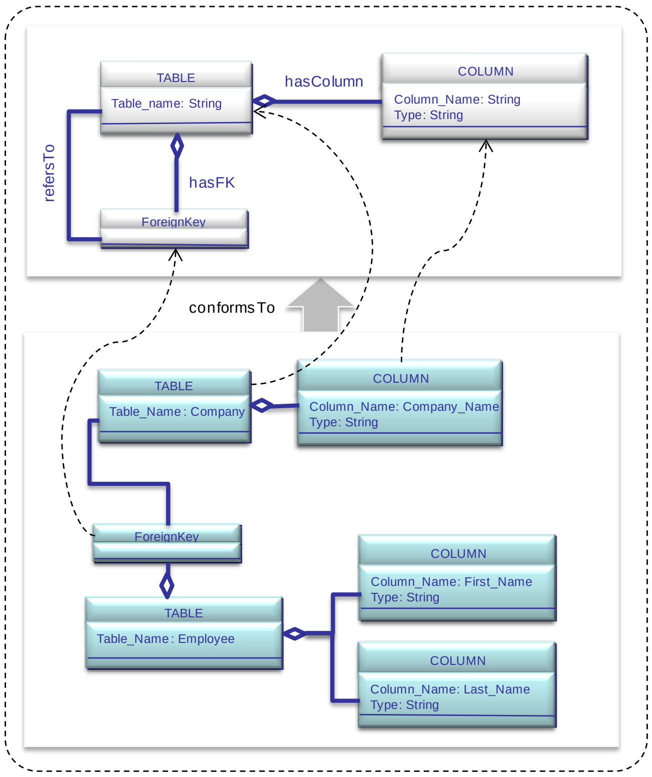


Figure 2-2. Simplified Relational Metamodel and Conforming model

2.1.4 Concrete Syntax, Abstract Syntax and Semantics

As we have already sketched, the first step towards the definition of a modelling language is the specification of its metamodel. In essence the metamodel collects the **abstract syntax** of the language, that describes the vocabulary of concepts provided by the language and how they may be combined to create models. On the other hand, the **concrete syntax** provide a notation that facilitates the presentation and construction of models or programs in the language.

There are two main types of concrete syntax typically used by languages: textual syntax and visual syntax. That is, a model can be expressed using a textual notation (like coding a program) or a graphical notation, the most common being diagrams and tree-like representations. In fact, you can define several concrete syntax for the same abstract syntax, i.e. the set of concepts collected in a particular modelling language might be expressed using several notations. For instance, you may opt for a nodes & edges notation to provide with an overview of the model and a textual one to provide with a more detailed view.

Along this dissertation the reader might find a distintinction in this line. Some times we will distinguish between a model and a **diagram**. The former will serve to refer to the abstract syntax of the model, while the later will refer to its concrete (visual) syntax.

By contrast, it is not so clear what the **semantics** of a modelling language is. If you have a look at [86] (a milestone in MDE literature) you will find clear and precise definitions on abstract and concrete syntax. However, they give a fuzzy definition on semantics. Whenever a MDE practitioner has tried to explain the term, he has resort to a classical state machine example to argue in favour of the need of defining the semantics of a language as a way of specifying how each meta-concept behaves when the model is executed. Another fuzzy way of thinking in the semantics of a la language is "that stuff about the (meta)concepts that you are not able to capture in the (meta)model" [25]. But even the theoreticians behind this statement makes a distinction between static and dynamic semantics and claim that the former is more or less collected in the abstract syntax while the latter is still to be addressed. Curiously, they conclude that the best way to express the semantics of a language is using the state machine abstraction.

In most cases, semantics are not explicitly define and they have to be derived from the run-time behaviour [111]. In our opinion, you can only define the semantics of an executable language, one that owns some dynamic component. For instance, a language to define state machines.

In fact, it had been one of the traditional differences between modelling and programming languages. Since the latter had to be executable, they needed a semantics definition for each (meta)concept. The other traditional difference had been associating visual notations with modelling languages and textual notations to programming languages. As we will show along this dissertation, these assertions are not valid in the current scene.

2.1.5 Metamodelling Frameworks

In essence, a metamodelling framework is an environment for the definition of models and metamodels [296].

To that end, it has to supply a precise meta-meta modelling language for the production of the abstract syntax of the modelling language, plus the mechanism to define its concrete syntax and generate the tooling for the new language [2]. That is, from the specification of the modelling language (both the abstract and concrete syntaxes), the framework has to be able to generate (or at least, provide the way to do it) complete editors for creation of models using the new modelling language. As well, it has to support models persistence and retrieveing.

Traditionally, these editors have been graphical editors following the nodes & edges style. However, the use of XML as underlying storage format has contributed on the rise of tree-like editors. Beside, note that according to previous sections, the concrete syntax might be expressed in terms of a textual or a visual notation. Hence, the generation of textual editors for modelling languages is also acceptable. In fact, it is gaining acceptance nowadays as a way towards easing the definition of very precise models.

2.1.6 Model Transformation

Even with a meta-model in hand, a graphical modelling tool is little better than a fancy drawing tool if we cannot automate the translation of these models into code, documentation or analysis.

Working with multiple, interrelated models requires significant effort to accomplish some tasks related with model management, such as refinement, consistency checking, refactoring, etc. Many of these activities can be performed as automated processes, which take one or more source models as input and produce one or more target models as output, following a set of transformation rules [356]. We refer to this process as **model transformation** [320].

In the MDE literature we can find several definitions of what model transformations are:

- The MDA Guide [246] gives a definition of model transformation: "Model transformation is the process of converting one model to another model of the same system".
- Kleppe et al. defines model transformation as "automatic generation of the target model from a source model, which conforms to the transformation definition" [193].
- Tratt uses the following definition: "A Model Transformation is a program which mutates one model into another; in other works, something akin to a compiler" [342]
- Sendall & Kozaczynski define model transformations as “Automated processes which take one or more source models as input and produce one or more target models as output, following a set of transformation rules” [320].

Figure 2-3 provides with an overview of the model transformation process.

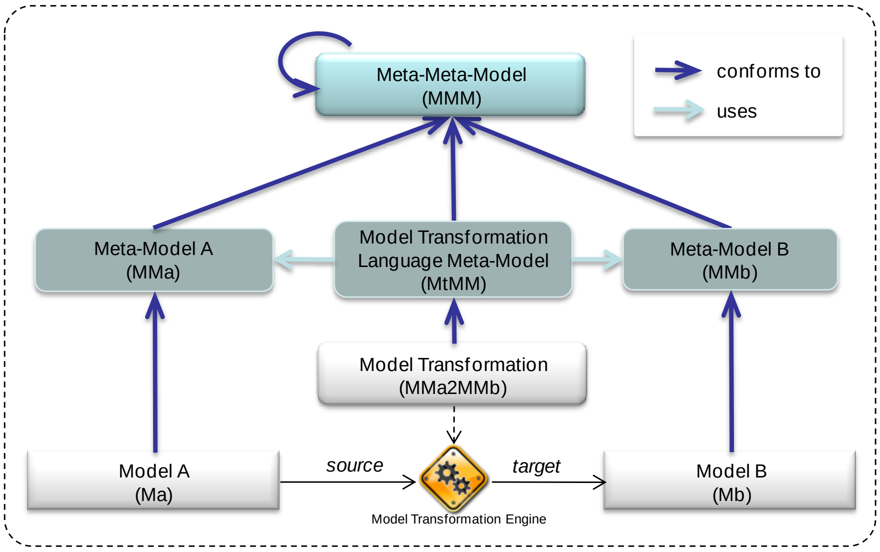


Figure 2-3. Overview of Model Transformation process

The root of the process is the metametamodel (MMM). It provides with a set of basic abstractions that allow defining new metamodels. Next, the source and target metamodels are defined by *instantiating* the abstractions provided by the metametamodel. They are said to **conform to** the metametamodel. Finally, the model transformation engine executes the MMa2MMb model transformation to map a model Ma into another model Mb. To do so, MMa2MMb specifies a set of rules that encodes the relationships between the elements from the MMa and MMb metamodels. The model transformation is defined at metamodel level, i.e., it maps elements from the input and output metamodels. So, it can be used to generate an output model from any set of models conforming to the input metamodel. In other words, the model transformation program works for any model defined according to the input metamodel.

Note that if we define the set of rules and constraints that drives the construction of a model transformation in a metamodel (MtMM), any model transformation can be expressed as a model conforming to that metamodel. Expressing model transformations as models (so-called transformation models) allow manipulating them by means of other transformations. This provides with several advantages. For instance, any model transformation can be the input or output of another model transformation. We use a specific term to refer to this type of model transformations. A **Higher Order Transformation** (HOT) is a special kind of model transformation whose input or/and output is a model transformation. In addition, we may compose transformation models like we compose any other type of models [51, 373], we can deploy metamodel evolution and model co-evolution techniques [84], define chains of model transformations [351], reuse exiting model transformations [309], etc.

In section 2.3, we provide with an overview on current model transformation approaches plus a brief overview on existing model transformation languages.

2.1.7 Weaving Models

Model transformation is essentially intended to define executable operations. Hence it is not always adapted to define and to capture various kinds of relationships between models elements. However, we often need to establish and handle these correspondences between the elements of different domains, each one defined by means of a model. The correspondences may be informal, incomplete, and preliminary. In many cases they may not be used directly to drive an executable operation. Model weaving is the process of representing, computing, and using these initial correspondences. This way, a set of correspondences between different model elements is represented as a **weaving model** [35].

A *Weaving Model* is thus a special kind of model used to establish and handle the links between models elements. This model stores the links (i.e., the relationships) between the elements of the (from now on) *woven* models. We illustrate this idea in Figure 2-4: Mw is a weaving model that captures the relationships between Ma and Mb (the *woven models*), denoted by the triple [Mw, Ma, Mb]. Then, each element of Mw links a set of elements of Ma with a set of elements of Mb. For instance, the r_2 element of Mw defines a relationship between a_2 and a_3 from Ma, and b_1 from Mb.

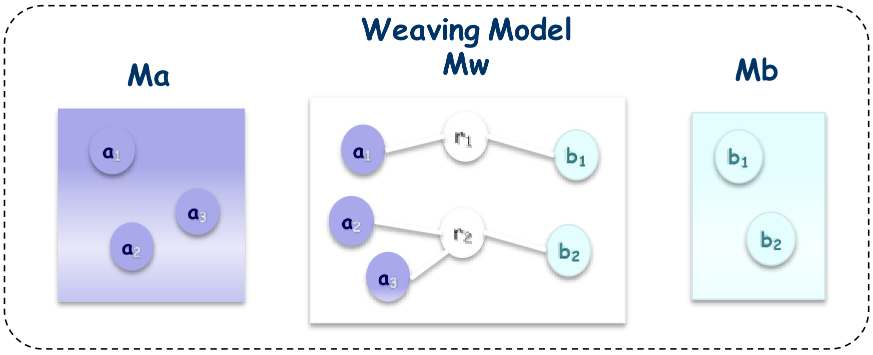


Figure 2-4. Model Weaving Overview

In the context of MDE, weaving models might help to implement separation of concerns [286]. Instead of using huge models comprising all the aspects of the system, it is more convenient to define a set of more manageable models, each one focused on modelling one aspect of the system. Them, the connections between those models can be specified using a weaving model.

In addition, the use of a weaving model to collect the relationships between the elements of source and target metamodels may help to develop the corresponding model transformation. Furthermore, if we express the relationships between the elements of two metamodels as a set of links contained in a weaving model, we are capable of using the information provided by that model to generate the program (the model transformation) that supports the transformation from one to another. For instance, in [356] we proposed a matching process based on cumulative weaving to generate automatically model transformations.

2.1.8 Code Generation

So far we have looked at software models as the plans of the system under development. However, we need a way to translate the model into the code that implements the system. This is done by **code generators**. In essence, the code generation process maps the specification of the system, collected in the software model, to a set of executables, undestable by execution platforms.

In some sense, code generation is similar to programs compilation where a tree-walker visits each node in the AST (Abstract Syntax Tree), to build smaller components and assemble them into larger components [140]. Likewise, a code generator navigates the model while it generates the code that implements each concept in the model.

A well-accepted approach is to collect all the infrastructure into a set of base classes and then have the user-viewed class inherit of them [140]. For instance, this is the case of the Eclipse Modelling Framework (see section 2.2.6), which provides with a complete framework of class libraries based on the model-view-controller deign pattern [148] for model handling, storing, editing, etc. Then, EMF generated code uses this classes directly and you may opt for using the *default* implementation, based on the definition of JAVA Interfaces, or modify it as needed by extending such interfaces.

2.1.9 Domain-Specific Modelling

Domain-specific languages (DSLs) are languages tailored to a specific application domain. They offer substantial gains in expressiveness and ease of use compared with general-purpose programming languages in their domain of application [241]. Indeed, like high-level programming languages rise abstraction higher from assembly language, DSLs rise abstraction another step higher.

So far, there is no paradigm that allows representing any type of problem in such a way that the solution could be automatically generated from the specification of the problem. Each domain has a set on inherent features that distinguish it from the others and that has to be considered when specifying the problem. Hence, there is a need for Domain Specific Modelling (DSM) [187] in order to provide with a vocabulary for each domain. This way, the problem could be expressed in a complete and reliable manner in order to automatically generate ist solution.

We can identify two different trends in the use of DSLs. On the one hand, we can find DSLs that support higher-level abstractions than general-purpose modelling languages, and are closer to the problem domain than to the implementation domain [299]. Such DSLs allow modelers to perceive themselves as working directly with domain concepts. On the other hand, we can make a simile between DSLs and programming languages. A DSL might be shown as the grammar of the language and a model expressed with such DSL would be a program written in the corresponding grammar. From such model, a **code**

generator automatically produces the code that implements the modelled program [138].

This is our point of view, we argue in favour of DSLs for platform-specific modelling, using them in a way similar to programming. That is, we rise one point the level of abstraction, good for programmers/developers but not enough for business analysts since the result is still too technical. For those non-IT stakeholders we provide with a higher abstraction level by means of platform-independent modelling. Then, we develop **model transformations** to translate their *human-understandable* designs to technical designs. In addition, since the translation or mapping process lend some space to decision-making when moving from the human to the IT technical space, we provide with a way to make them in the form of **annotations**. Note that these annotations are to be made by the developers, probably assisted by the business analysts, since they do not have to be aware of technical concerns. Finally,

It is worth mentioning that, though there is a trend towards using textual syntaxes for Domain Specific Modelling (DSM) [187], the authors were thinking mainly in graphical modelling languages when they invent the term. In Steven Kelly's words [190] "when we were looking for a name (…) DSL was clearly similar, but we needed to distinguish what was different: we were talking about graphical modelling languages". However, DSM is not just about attaching visual syntaxes to programming languages. This is mainly related with external DSLs. But you might use a DSL also to rise the abstraction level when coding, or at least, to ease the task [138].

2.1.10 What Model-Driven Engineering is

After reviewing the concepts that forms the basis of MDE, here we focus on explaining how they combine to constitute a new software development paradigm.

So far, Software Engineering has been tradidionally identified with programming tasks, what partially explains why it has been conceived as a minor discipline when compared with other engineering disciplines. The rest of engineers, like civil, agronomist, mining or aeronautic are responsible for making a design of the system prior to its construction. In fact, they are not the ones that effectively build the system. They just supervise the development process. A similar approach has to be brought to software engineering. To that end, there is a need for raising the abstraction level.

The Model Driven Engineering (MDE) paradigm [41, 131, 143, 166, 316, 328, 368] is a new trend in software engineering whose main proposal is to focus on models rather than in computer programs. MDE is a natural step in the historical tendency of software engineering towards raising the abstraction level at which software is designed and developed. Assembly languages gave way to structured programming languages that yielded to object-orientation and so on.

Appart from raising the level of **abstraction,** MDE aims at increasing the level of **automation** in software development. To that purpose, the idea promoted by MDE is using models that specify the software system at different levels of abstraction. This way, higher-level models are transformed into lower-level models until the model can be made executable using either code generation or model interpretation. The increase of automation comes mainly from the fact that the step from one model to the following is performed by using executable model transformations.

MDE has been applied in different contexts, resulting in a vast amount of model-driven methodologies for software development that covers almost every field of software engineering, from Web Engineering to real-time systems, database development, etc. All these proposals consist of a development process, a set of (meta-)models handled along that process and a set of mappings between them. The mappings between models play a very important role since the process proposed is always a continuous development process, which according to the MDE principles consider the models as the prime actors. As Figure 2-5 shows, each step of this common process consists basically on the generation of an output model starting from one or more input models over which the mapping rules are applied. In the remaining steps of the process, this output model acts as one of the input models. Therefore, the process could be summed up as the sequence of model transformations that have to be carried out in order to obtain the different models defined in the process, until the last one, that is, the working code, is generated − notice that the working code is no more than another model, this one with the lower abstraction level.

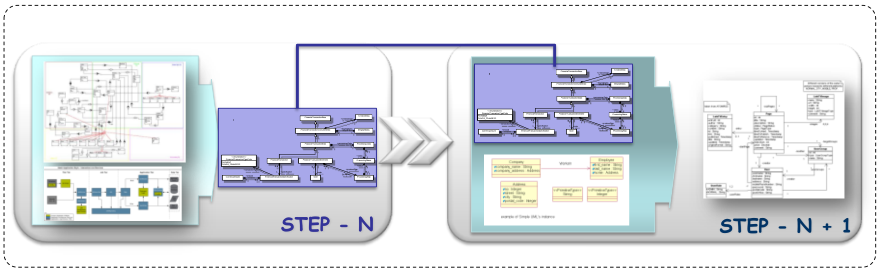


Figure 2-5. Simplified overview of MDE

2.1.11 Model-Driven Architecture

The principles of MDE emerged as a generalization of the Model Driven Architecture (MDA) [143, 157, 193, 238, 239, 246], proposed in 2001 by the Object Management Group (OMG). Indeed, Favre [131] states that MDA is a specific encarnation of MDE.

MDA is a framework for software development aligned with MDE, whose main characteristics are the definition of models as first class elements for the design and implementation of systems, and the definition of mappings between those models, which allow such transformations to be automated.

MDA considers three big groups of models according to its abstraction level. System requirements are modelled by Computer Independent Models (CIMs). Platform Independent Models (PIMs) allow modelling system functionality, without taking into account any specific platform. Finally, specifications described in the PIMs are adapted to the specific platforms by means of Platform Specific Models (PSMs) from which the code is automatically generated.

Besides, the OMG proposes a set of standards to put MDA to work (some of them, like the UML existed before the advent of MDA). In fact, some recognised authors argue that MDA is MDE with OMG standards [139].

In the following, we summarize them since they are probably the main contribution of OMG to MDE, apart from the definition of the different abstraction levels.

2.1.11.1 MOF

The *Meta-Object Facility* (MOF) [265] serves as the metadata management foundation for MDA. MOF provides a standard for specifying metamodels, i.e. a meta-metamodel, which is the root of the metamodelling hierarchy shown in Figure 2-6. MOF is defined at level M3 and serves to define models at M2 level.

Note also that MOF is reflective, thus it is defined in terms of MOF itself. Next, the models at M2 are theirselves metamodels for the models defined at M1 that still own some level of abstraction regarding the objects layer situated at M0.

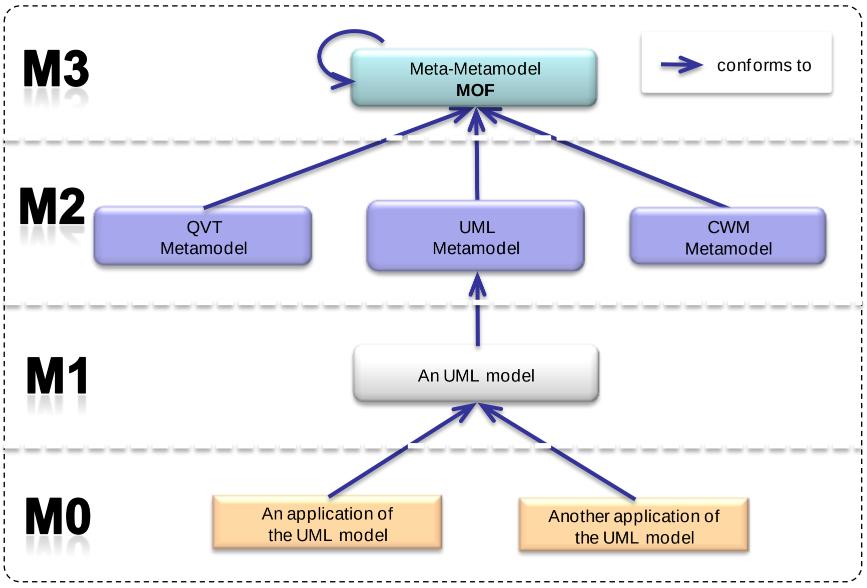


Figure 2-6. OMG four layered metamodel architecture

The relevant part of this architecture is the ability to navigate from one element (it does not matter if it is a class, an object or whatever) to its corresponding metaobject. This is shown in Figure 2-7. At M3 level, the MOF specification states that any model conforming to MOF is composed of two types of objects: classes and associations. At M2 level, both types of constructions might be used to define any desired model conforming to these statements. For instance, the UML metamodel is in its turn a MOF conforming model. It states that any UML model will contain classes, which in turn contain properties. This is stated by the composition association that connect both metaobjects. At M1 level, a simple UML model contains one class (*Customer*) that contains one property (*name*). Finally, the UML model admits infinite instantiations, one particular example is shown at level M0.

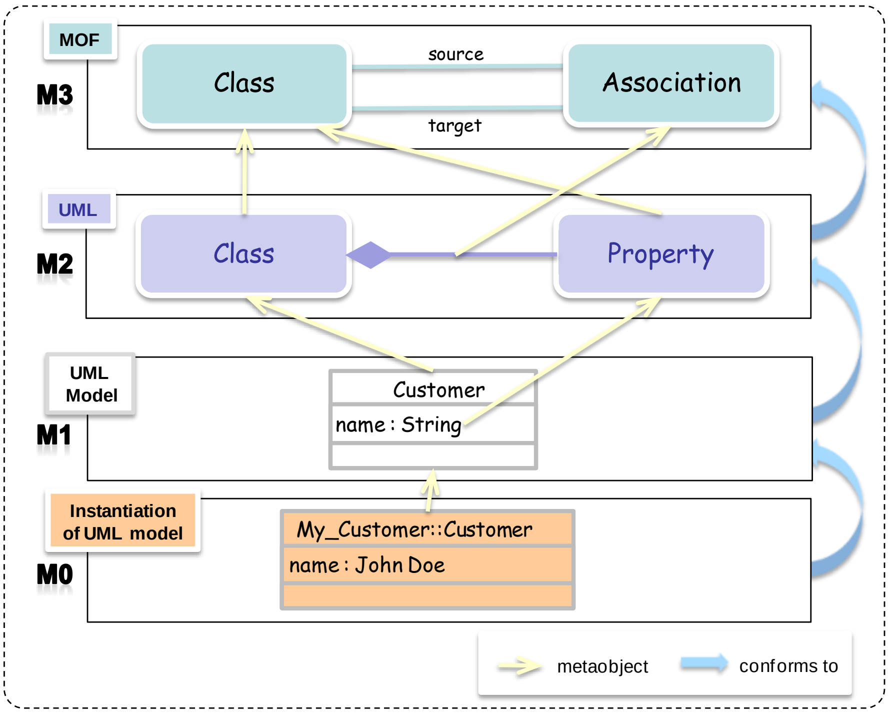
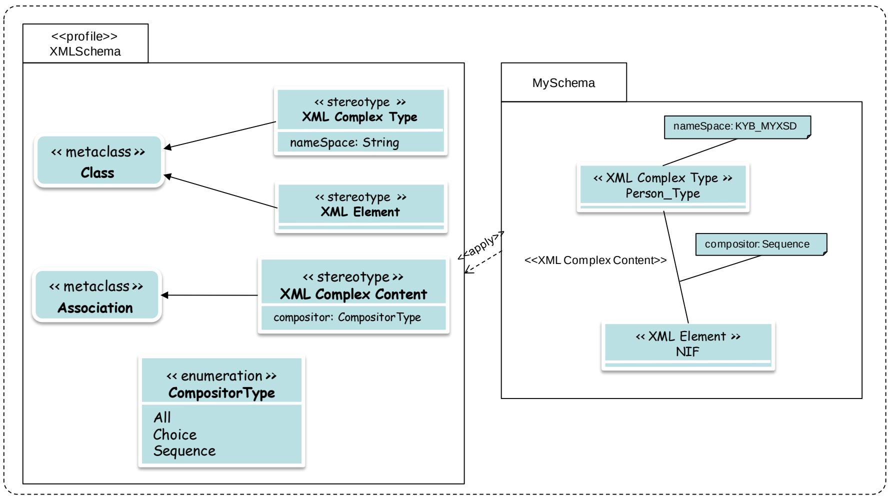


Figure 2-7. Applying the OMG four layered metamodel architecture

2.1.11.2 UML

The Unified Modelling Language (UML, [270]) is a widely recognised and adopted modelling language. It is a general-purpose language (GPL) specially intended for modelling object-oriented software systems. The basic building block of UML is a diagram. There are several types of diagrams for specific purposes (e.g., time diagrams) and a few for generic use (e.g., class diagrams).

Besides, it defines a lightweight extension mechanism so-called UML profile. A UML profile is a modelling package containing modelling elements customized for a specific purpose or domain. It combines stereotypes, tagged values, and constraints in order to define a variation of UML for a specific purpose. In other words, a UML profile defines new types of modelling elements by extending existing ones. For instance, the (simplistic) UML profile for XML Schema modelling shown at left-hand side of Figure 2-8 provides with three new types of modelling elements: the *XML Complex Type* and *XML Element* classes and the *Complex Content* association. Besides, modelling the type of compositor used to define the Complex Content of the Complex Type is supported by adding a *Compositor* tagged value. Likewise, the namespace of the Complex Type is modelled with the *nameSpace* tagged value. The result may be considered as a new metamodel at M2 level that can be used to define new models at M1 level, like the one shown at left-hand side of Figure 2-8. It models an XML Schema containing just a Complex Type (*Person_Type*) that in turn contains one XML Element (*NIF*).



<<profile>> XMLSchema
« stereotype » XML Complex Type
nameSpace: String
« metaclass » Class
« stereotype » XML Element
« metaclass » Association
« stereotype » XML Complex Content
compositor: CompositorType
« enumeration » CompositorType
All
Choice
Sequence
<<apply>>
MySchema
nameSpace: KYB_MYXSD
« XML Complex Type » Person_Type
compositor: Sequence
<<XML Complex Content>>
« XML Element » NIF

Figure 2-8. Defining and using an UML profile.

2.1.11.3 XMI

XML Metadata Interchange (XMI, [275]) is an XML-based standard for sharing meta-data. It can be used to represent ordinary data as well. That is, XMI might be used for both serializing objects in XML documents and to generate schemas from models.

Thereby, XMI includes several artefacts. The most relevant being:

- A set of rules to generate XML Schemas for MOF based metamodels.
- A Schema for UML
- A Schema for MOF
- A set of rules to generate XML documents from instances of MOF models.

In fact, since UML is the most popular MOF model, the XMI Schema for UML (also known as XMI[UML]) has been the most commonly adopted. Figure 2-9 shows the relation between UML and XMI. Any MOF model, like the UML metamodel, is persisted in a XMI Schema. Likewise, any UML model is persisted in an XML document conforming to the above Schema.

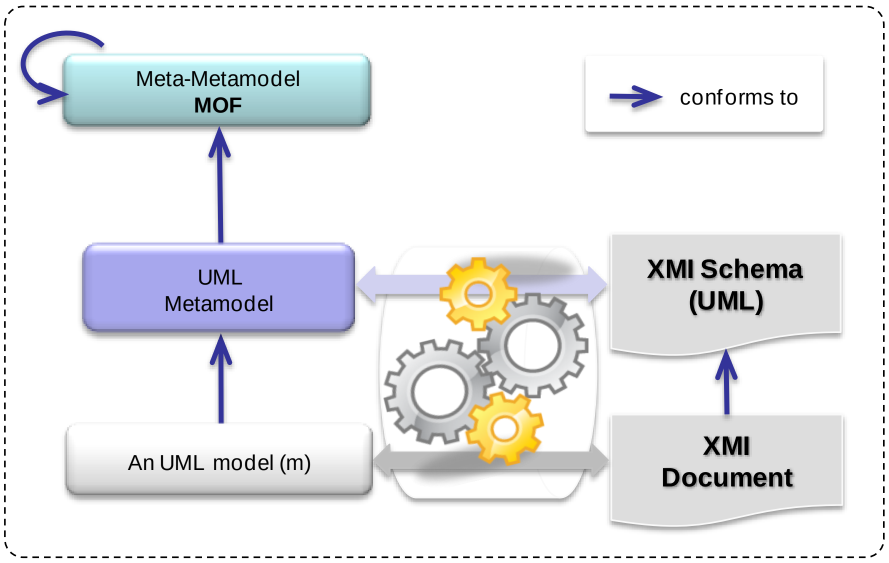


Figure 2-9. Using XMI

Unfortunately, nowadays, the XMI documents generated by most tools present some differences from the XMI standard and are seldom interchangeable [70]. Indeed, although it is widely accepted as storage format, the whole point of XMI is interoperability. It is quite rare to find two different tools using the same XMI version [188].

The problem is well-stated by Uhl in [344]: "(…) XMI can come in handy for integration if you are lucky enough to find two compliant tools with matching XMI versions and metamodels or, alternatively, if you are XSLT-literate. Of course, exchange is limited to the model's abstract syntax because Gentleware, the one company that participated in the UML diagram interchange standard, remains the only supporter".

In the end, the only way of solving the interoperability problem that has proved itself to be useful and efficient is the use of model transformations [55, 113].

2.1.11.4 OCL

The Object-Constraint Language (OCL, [268]) is a declarative language to define expressions that apply to any MOF conforming model, like the UML metamodel, or any UML conforming model. Though it was conceived as a constraint definition language, OCL might be considered as a general-purpose query language. Indeed, the most of the existing model transformation engines use OCL as navigation language or, at least, an OCL-like language.

Besides, OCL allows defining invariants, that must hold for the model elements and pre- and post-conditions for on actions or operations. Working this way, OCL allows adding value to any model.

For instance, Figure 2-10 shows a simple class diagram capturing a (simplistic) mortgage system taken from [374].

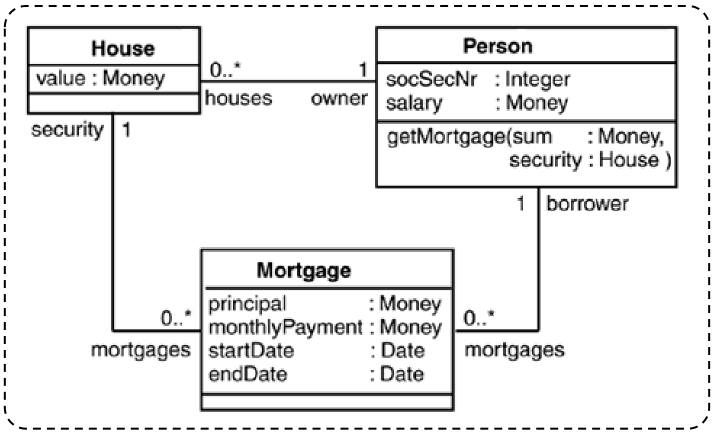


Figure 2-10. The mortgage system expressed in a class diagram

Needles to say, there are some rules that the diagram itself is not capable of capturing. These are collected in a set of OCL expressions in next to its description in natural language.

Table 2-1. OCL Expressions for the mortgage system diagram [374]

OCL Expression	Meaning
context Mortgage inv: security.owner = borrower	A person may have a mortgage on a house only if that house is owned by him- or herself; one cannot obtain a mortgage on the house of one's neighbor or friend
context Mortgage inv: startDate < endDate	The start date for any mortgage must be before the end date.
context Person inv: Person::allInstances()->isUnique(socSecNr)	The social security number of all persons must be unique.

2.1.11.5 QVT

The MOF Query/View/Transformation (QVT, [273]) is the OMG standard for model transformations. It allows defining three types of constructions:

- A Query is an expression evaluated over a model that results in a set of objects fulfilling the restriction imposed by the query. It results in one or more instances of types defined in the source model, or defined by the query language. Indeed, OCL is an example of a query language and is actually used in QVT.
- A view is a model which is completely derived from another model (the base model). A view cannot be modified separately from the model from which it is derived. Changes to the base model cause corresponding changes to the view. A query is a restricted kind of view
- Finally, a transformation generates a target model from a source model. Transformations take a model as input and update it or create a new model. In fact, a view is a restricted kind of transformation in which the target model cannot be modified independently of the source model. If a view is editable, the corresponding transformation must be bidirectional in order to reflect the changes back to the source model.

The QVT specification provides with three different languages:

- The QVT Relations is a declarative transformation language that allows defining the relations that must hold between the elements of the source and target model.
- The QVT Core language is a simpler though equally expressive declarative language. Indeed, the Relations language might be expressed in terms of the Core language but the latter does not provide with automatic traceability support.
- Finally, the QVT Operational-Mapping language is an imperative language that may be used to extend the Relations language with imperative constructions.

Due to the relevance of model transformations as thriving force of any model-driven development process, QVT exiting implementations will be widely covered in section 2.3.3.11.

2.1.12 Eclipse

We could look at Eclipse as a framework to build Integrated Development Environments (IDEs), i.e. it is an IDE whose architecture was designed to be used as underlying infrastructure to develop new IDEs for new languages or models. To that end, its infrastructure provides with extension points to *plug* the new IDEs, so-called plug-ins, in the Eclipse IDE. Because of its extensibility and industrial-quality user interface, Eclipse has been widely adopted as underlying platform for every type of sotware engineering supporting tools.

This way, Eclipse platform is a kernel so-called Platform Runtime plus a set of plug-ins. Apart from such kernel, everything in Eclipse is a plug-in. The Platform Runtime is in charge of discovering, loading and executing the different available plug-ins at run-time. Once Eclipse is launched, the platform runtime offers an integrated IDE composed by the available plug-ins. Thus, Eclipse is a kind of puzzle where each plug-in constitutes a piece that is identified and assembled with the rest. New functionalities are encoded in new plug-ins that connects with the existing ones.

Though all the plug-ins are treated equivalent, it is worth to pay special attention on two of them shown in Figure 2-11:

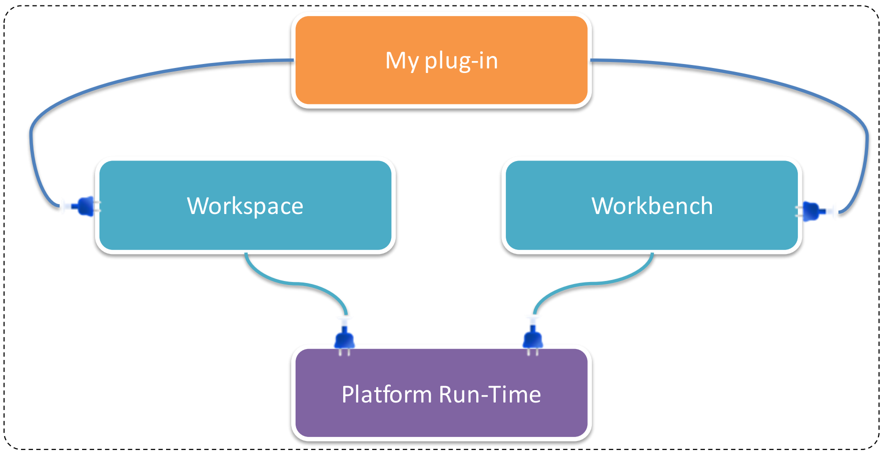


Figure 2-11. Eclipse Workspace and Workbench plug-ins

- The user interface rests on the **Workbench** plug-in. It defines a series of extension-points to extend the user interface as needed. For instance, we may add new toolbars; create new views (a view defines the layout of the workbench) or record notification requests for any event. If you need to build an XML editor, you will develop a plug-in that extends the classes that implements the Workspace. This way, you might color the text in a special way and so on.
- On the other hand, the **Workspace** supports the managemet of the resources displayed on the IDE, such as projects, folders or individual files. The workspace contains a number of projects that has a one-to-one correspondence with a folder on the operating file system. Each resource is

represented as an object of the workspace. Thus, there are classes to abstract them and one can extend and customize those classes in order to get files, folders or projects with special features.

The different Eclipse plug-ins are grouped into Eclipse projects, that collects all the (sub-)projects focused on development of tools focused on the same topics or purposes. In the following we give a brief overview on the Eclipse Modelling Project, that aims at grouping together all the MDE technologies developed atop of Eclipse.

2.1.12.1 The Eclipse Modelling Project

In general, organizations that adopt standards do not use to follow the standard as-is. The preferred way of working is to follow the reference implementation. Within the scope of MDE, the OMG standards presented so far provide the language definitions needed to achieve the goals of a metamodelling framework. However, the OMG does not provide a reference implementation and when looking into the details, the language definitions are partially incomplete, inaccurate, or ambiguous.

In absence of such reference implementation, practitioners tend to agree on a de facto standard upon which their proposals are built. Eclipse, and more specifically, EMF has been playing this role in the context of MDE. The advent of the Eclipse modelling Project (EMP) has contributed to bridge the gap between different modelling tools by providing a set of frameworks, tools and reference implementations for standards that help on the development of support for any methodological proposal based on MDE principles.

The Eclipse Modelling project is logically organized into projects that provide the following capabilities: abstract syntax development, concrete syntax development, model-to-model transformation, and model-to-text transformation. A single project, the Model Development Tools (MDT) project, is dedicated to the support of industry-standard models. Another project within the Modelling project focuses on research in generative modelling technologies.

Figure 2-12 gives an overview of the structure of the modelling project and its functional areas taken from [161]. As you can see, the Eclipse Modelling Framework is at the center. It provides with abstract syntax-development capabilities. EMF Query, Validation, and Transformation complement the EMF core functionality. Teneo and CDO provides with database persistence of model instances. Surrounding the abstract syntax-development components are model-transformation technologies. They include both model-to-text (Java Emitter Templates [JET] and Xpand) and model-to-model (QVT and ATL). However, notice that not all the solution for model-to-model and model-to-text are enclosed in the “official” Eclipse projects. VIATRA and MOFScript are examples of this fact. Concrete syntax development can be implemented in the form of graphical editors or textual editors. To that end, you might use the Graphical Modelling Framework (GMF) and the Textual Modelling Framework (TMF) respectively. Finally, a series of orbiting projects and components represent models, capabilities, and research initiatives available from the Modelling project.

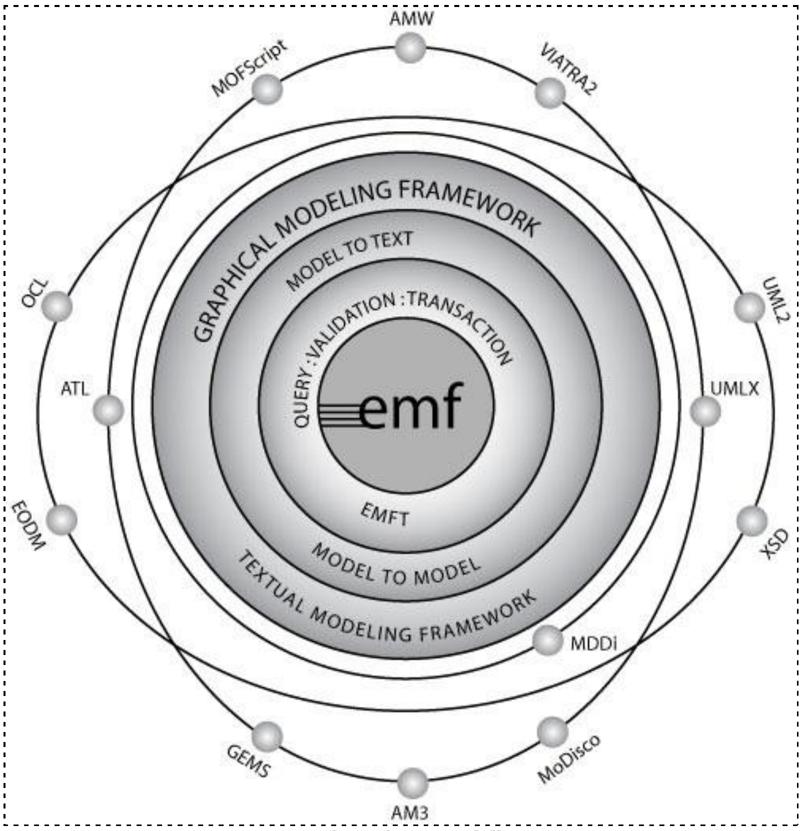


Figure 2-12. The Eclipse Modelling Project

2.1.12.2 The Eclipse Modelling Framework

As we just mentioned, all the facilities provided by the EMP are built on top of a common basis, the Eclipse Modelling Framework (EMF) [66, 161], that provides with the utils needed to define, edit and handle (meta-)models. Indeed, the strength of EMF has given rise to a new generation of EMF tools during the last years. We will present some of them in forthcoming sections. We will itroduce EMF capabilities and how are they to be used for building M2DAT's modules along this dissertation.

After this brief overview of MDE terms, we address the main part of this Chapter: the State of the Art on technology to support model-driven development of software.

2.2 Tools supporting MDE tasks

As stated in the introduction of this chapter, the emergence of MDE has resulted in the advent of a wide set of tools for supporting MDE tasks. Some of them provides with specific capabilities, mainly modelling and metamodelling functionality. Others try to integrate all the needed functionality, adding model transformation engines, code generators and the like. This section studies those works focusing on exiting metamodelling frameworks and paying a special attention on EMF, which has contributed decisively to boost the MDE paradigm [52]. Besides, some examples of tools providing with individual capabilities are cited.

These tools are grouped together because, in terms of MDE, they are not general-purpose tools, since they focus on supporting a special functionality, such as developing textual editors. However, all of them are valid for any DSL. In contrast with M2DAT, that is built to deal with the set of DSLs proposed in MIDAS methodology, the following tools are not devised to work with a particular DSL. They are used to build domain-specific tools. In other words, this section aims at reviewing the technology that we might use to build M2DAT. The objective is to be ready to face the selection of technology that will drive the specification of M2DAT.

Likewise, it is worth mentioning that we will not reference explicitly diagrammers or pure UML tools like magicDraw, Fujaba, IBM Rational, etc. for two interrelated reasons:

- We are not interested in working (just) with UML. In M2DAT, UML will be used with platform-independent modelling purposes, but UML models will have to be mapped to M2DAT DSL models. The proprietary storage formats used by those tools, as well as the classical XMI versioning problems (see section 2.1.11.3) advise against the use of these tools in MDE processes where different DSLs/metamodels are to be interconnected.
- In addition, we want M2DAT models to be, not only translated into code, but also validated, weaved, edited with different editors, etc. In particular, we

want to be able to handle M2DAT models with other existing and forthcoming tools. Thus, building M2DAT models with the above-mentioned type of tools is completely discouraged.

Note also that the aim of this thesis is the specification of an open-source extensible framework for model-driven development of WIS. Thus, building it in top of a commercial tool makes no sense. Therefore, we will limit to mention some of the existing commercial tools, but we will not consider them to build M2DAT.

Finally, before presenting each reviewed tool, we sketch the evaluation criteria used to asses them. Examples of evaluation criteria can be found on existing literature on software engineering [54, 303]. Likewise, there are more recent focused on evaluating MDE proposals, like [97], focused on Model Transformation, [99] focused on Bidirectional Transformations, or [361] focused on Graph Transformations. Here, we have defined one that is structured according to our needs to build M2DAT.

2.2.1 Evaluation Criteria

We need to have effective criteria to compare existent tools for MDE tasks. To that end, we would like to evaluate each tool in relation with the follwing set of features (next to each future, we state the possible values):

- **Scope.** [Values: Commercial / Academic / Open-source]

  One of the main concerns regarding software engineering tools is whether they are commercial or open-source tools. Since we want M2DAT to be an open-source tool, there is no sense in using any commercial component to build it. This way, the first criteria to discard existing tools for MDE tasks will be their scope. In addition, we will mention if it is an academic tool, just to provide with some more information.

- **Metamodelling.** [Values: YES / NO].

  The first step towards a new MDE methodological proposal is the definition of a new modelling language (whether it is a UML profile or a DSL). Therefore, as part of building M2DAT we need tools supporting the definition of new metamodels (the metamodel defines the abstract syntax for the new language, we will talk later about supporting the concrete syntax). Besides, those tools should provide with the tooling to "instantiate" the metamodel, i.e. to define terminal models that conform to the new metamodel. Therefore, we need to check which of the reviewed tools offer metamodelling support and how does it work in order to choose one for M2DAT.

- **Model-to-Model Transformations**. [Values: YES / NO / Limited]

  We have alredy mentioned that model transformations are the cornerstone to support MDE proposals. In particular, the holy gray of automation is not feasible without model transformation support. Thus, when evaluating components to build M2DAT, we need to study whether those components include facilities to develop model-to-model transformations.

- **Model-to-Text Transformations**. [Values: YES / NO / Limited]
- This feature complements the previous one. Here we study if the evaluated component does provide support for model to text transformations, i.e. code generation capabilities. Besides, we will show that there are a number of components exclusively focused on model-to-text transformations, like AndroMDA or the MDWorkbench..
- **Validation.** [Values: YES / NO / Limited]

  A common issue related with MDE tools is the support of models validation [91]. In spite of the proliferation of methodologies and tools for MDSD, we have detected that most of them do not include activities and/or features related to the analysis of the constructed models built or, if they exist, they are rather weak. These activities are especially important in proposals aligned with MDE since models are used as the mechanism to carry out the whole software development process. Thus, errors at initial stages of development will be reproduced in the subsequent generated code [249]. This can be avoided by providing support to specify constraints at metamodel level and to evaluate then on terminal models. We aim at integrating model validation mechanisms in M2DAT. Therefore, we will study whether each reviewed tool supports this feature and how it is done.

- **Graphical Editors**. [Values: YES / NO / Graphical / Textual]

  This feature might be stated as supporting the definition oa graphical concrete syntax for a new modelling languages. A common issue related with MDE tools is usability. To enhance usability, the tool has to provide with graphical editors to edit terminal models conforming to previously defined metamodels. Therefore, we will analyse the support of each tool (if existing) to develop graphical editors for terminal models. We will not limit to boxes and arrows editors, we also refer to tree-like editors, like the ones from EMF, or any other graphical way of defining models. To summarize, here we will study if the tool is able to generate an environment for handling models conforming to a given metamodel.

- **Standardized**. [Values: YES / NO].

  Here, we are interested on analysing to which extent the tool is aligned with standards. When we talk about standards in MDE contexts, we are mainly refererring to OMG standards, like UML, MOF or OCL. For instance, we will consider if the metamodelling capabilities are based on MOF [265], the model-to-model transformation is based on QVT [273] or the validation mechanisms based on OCL [268]. It is worth mentioning that we will consider not only standards *de jure*, like MOF, but also its reference implementations (in case they exist). This way, since Ecore [66] is considered the *de facto* standard for metamodelling, we will consider a tool based on Ecore as an standardized tool.

- **Extensibility**. [Values: YES / NO / Partially].

  Another key issue for us when designing M2DAT is reaching the highest level of extensibility. We aim at integrating in M2DAT any interesting technical solution for MDE that arises. To that end, we need to build M2DAT in top of components that can be extended. Therefore, we will analyse how easy it results to integrate new functionalities into reviewed tools. Notice that a number of tools claim to be extendable but when you address the task of developing the corresponding extension, you realise that it is a rather challenging task. In this sense, the perfect example of extensible framework is Eclipse, which was specifically devised to be extended.

- **Interoperability**. [Values: YES / NO / Partially].

  This point is directly related with the previous one. Since we aim at using M2DAT as a test bench for any new appearance in the field of MDE components for development of MDE tools, we need it to be highly interoperable with other tools. So, it has to be built on top of components that provides with automatic import/export mechanisms for software artefacts developed with other tools. At worst, we need tools for which building support for migration of software artefacts can be developed in reasonable time and manner. The major advance in terms of interoperability for MDE tools in recent years was the advent of EMF. Since it provides with an underlying model management framework on top of which MDE tools could be developed. The rest of tools developed on top of EMF handle models developed with such tools with no additional effort. We could say tha EMF is the “esperanto” of MDE tools. Therefore, any tool developed on top of EMF will be judged as highly interoperable. However, running atop of EMF is not the only way to achieve interoperability. For instance, purely UML-based tools that use XMI should be also interoperable.

- **EMF-based.** [Values: YES / NO / TO-DO (it is planned)].

  The previous point has clarified why we are interested in evaluating the level of compliance of any tool for MDE tasks with the EMF framework.

In the following section, we review the main tools for MDE tasks according to the above-described features that compose our evaluation criteria.

2.2.2 AndroMDA

AndroMDA [17] is a template-based code generator framework from UML models for J2EE, Spring and .NET platforms. The functionality to provide source code for a specific platform is collected on a cartridge. A set of cartridges oriented to the current development kits, like Axis, jBPM, Struts, JSF, Spring or Hibernate is included by default. In addition, you can develop your own cartridge or modify an existing one by extending a generic cartridge so-called *Meta*.

Current release, AndroMDA 3.3, is an open-source and stand-alone tool. At the beginning of 2007, the authors started to work on a new release (AndroMDA4) to be integrated into the Eclipse platform. It added metamodelling capabilities, plus model transformation support (using the ATL language [387]) and visitor-based code generation (using the MOFScript language [391]). This is why we place AndroMDA in this category instead of plaging it just in the model-to-text transformation engines (section 2.3.4). However, AndroMDA4 is on hold and very experimental since the developer behind (Matthias Bohlen) shifted his focus to other activities.

To sum up, we may qualify AndroMDA as an open-source framework for model-to-text transformations. It does not provide support for metamodelling, model-to-model transformations, model validation or model management. On the other hand, it is highly extensible since you might develop your own templates. Besides, it is rather interoperable since works with UML models, though the XMI versioning problem (see 2.1.11.3) might complex real interoperability. Finally, EMF compliance is planned, though not supported at the time of writing this dissertation.

2.2.3 ATOM^3

ATOM^3 (A Tool for Multi-formalism and Meta-Modelling, [107]) is a framework for the definition of multi-view languages that incorporates mechanisms for syntactic and semantic validation, as well as metrics to evaluate the quality of a design and trigger re-designs when needed.

Although it is mostly known as a graph-based model transformation framework, it does provide metamodelling support plus validation facilities. However, the rules that drive this validation have to be coded as preconditions in Python, the underlying language or ATOM^3.

We qualify it as not standardized since it does not follow any OMG standard. In addition, it owns a low interoperability level since ATOM^3 models does not use any common underlying format. Finally, nothing is said in its documentation about the ability of adding/modifying its functionality.

Theoretical foundations of ATOM^3 make it a very appealing tool. However, it seems not to be ready for production settings yet. Some parts of the documentation are outdated and even the basic example caused several exceptions and useless warnings. Nevertheless, it is one of the few tools providing with real graph-based model transformations.

Regarding evaluated features, ATOM^3 is an academic tool that allows defining new metamodels using the E/R model [82]. From such metamodel, it provides with a basic graphical editor for conforming models. Besides, it supports model-to-model but not model-to-text transformation. As mentioned, model validation is also supported, though not in a user-friendly manner. It does not follow any of the OMG standards and no information is provided about extension capabilities. Besides, nothing is said about the ability of importing/exporting model to/from other tools. Finally, it is not EMF-compliant.

2.2.4 DOME

Domain Modelling Environment (DOME) is an extensible system for graphically developing, analyzing and transforming models of systems and software [129]. That is, DOME is a metamodelling framework.

It aims at providing toolsets for newly defined metamodels. To that end, you have to define a notation using a meta-tooling model called DOME Tool Specification (DTS) that includes a set of predefined constructions, such as model, graph, component, port, etc. From that notation, DOME generates the code that implements the desired toolset. DOME provides just basic support for the definition of the visual concrete syntax.

The definition of restrictions is based on the use of a scripting language called Alter. Besides, Alter is said to be a way to code model-to-model and model-to-text transformations in DOME.

In the beginning, the code generated was SmallTalk, the code in which DOME was developed. From 2003, the whole framework is been re-implemented in JAVA to support also JAVA generation. Likewise, they plan to add Eclipse-integration capabilities.

It is worth mentioning that at the moment of writing this dissertation DOME seems to be abandoned. So far, all the references that we have found pointing to DOME have turned out to be dangling references

All things considered, DOME supports metamodelling and edition of conforming models, plus (limited) model-to-text transformations and (limited) model validation. Nevertheless, coding of complex transformations using the Alter language is not feasible and there seems to be no way of connecting DOME with existing model transformation engines. Besides, it does not conform to any standard and it is not extensible neither interoperable with other tools.

2.2.5 DSL Tools

The DSL Tools from Microsoft [90] is a suite for creating, editing, visualizing, and using domain-specific models.

To that end, a graphical editor is used to create a domain model using a set of predefined constructions. From such model, a graphical editor for confoming models is automatically generated. In addition, it allows defining code generation templates that takes as input such terminal models.

Nevertheless, there is no support for model-to-model transformation neither for defining constraints over terminal models. The standardization level is is nil since it is completely based on proprietary notations and there is no way of extending the platform.

2.2.6 Eclipse Modelling Framework

We cannot state that the Eclipse Modelling Framework (EMF, [161]) does or does not provide with specific capabilities, like model-to-model transformation or code generation.

Actually, EMF itself does not provide with these facilities, but as we have already shown in section 2.1.12.1, the EMP projects collect all these facilities. Indeed, those projects will be presented as isolated components in the following sections, since we might use EMF with or without each one of those components to build M2DAT.

All this given, regarding the target features to evaluate, we can state that EMF is a metamodelling framework, devised to be extended and providing the highest interoperability level.

In section 2.1.12.2 we will provided with a detailed insight in EMF principles since it deserves a special attention due to its widespread adoption as underlying model management framework. So far, we have just focused on how EMF behaves regarding the features pointed out in section 2.2.1.

2.2.7 EMFATIC

Emfatic is a language for defining Ecore models [172]. It uses a compact and human-readable syntax similar to Java. The Emfatic plug-ins supplies an editor and a parser for the language. They support actions to compile Emfatic source code into an Ecore model and allow Ecore models to be decompiled into Emfatic source code (injection/extraction). Emfatic itself builds upon Gymnastic [153], a framework for jumpstarting text editors for custom Domain Specific Languages.

Emfatic's main functionality is injection/extraction of Ecore from/to textual specifications. Thus, you can use Emfatic with metamodelling purposes. Instead of defining en Ecore model, you may prefer defining your metamodel using the Emfatic language. In fact, some authors argue in favour of textual editors for DSL against graphical editors [183]. We bet for combining both approaches [368].

Regarding evaluated features, Emfatic just provide with metamodelling capabilities and generates a textual editor for conforming models. Thus, no support for model transformations, model validation of graphical edition of models is provided in Emfatic. Besides, we may qualify it as rather standardized and interoperable since it is completely based on EMF.

2.2.8 GME

The Generic Modelling Environment (GME, [103, 216]) is a mature and recognised metamodelling framework that was born before the boom of MDE.

GME supports its own metamodelling language to define metamodels (*Paradigms* in GME jargon) so-called MetaGME. It is a subset of UML that includes abstractions like Atom (any model element), Model, Connection (association), Attribute, etc. Visual syntax is defined by attaching *decorator* objects to the concepts included in the metamodel. From the metamodel and the decorators' specification, GME generates an editor for conforming models that provides with several functionalities, like zooming, undo/redo, etc. Additional constraints to be checked over terminal models can be added using its own flavour of OCL.

A very interest feature is the ability to register several versions of the same metamodel. In some sense, this mitigates the problem of metamodel evolution and model co-evolution [84]. Model transformations can be attached in GME, by they have to be developed using C++.

Since GME is based in MS COM, it can be extended using any language that supports COM, primarily C++ and Visual Basic. Model transformations can be developed for GME models using the GReAT language [9] that will be introduced later. Besides, the GReAT language provides with some limited capabilities to translate models to code but we would not say it supports properly code generation.

Finally, it is worth mentioning some works that have focused on bridging GME and EMF:

In [45] the authors use the AMMA (ATLAS Model Management Platform) tools to that end. Apart from some development problems, like the loss of graphical data from GME models when they are carried to the EMF world, the authors point out the complexity of the task. They claim that more advanced MDE frameworks were needed for this task.

Besides, the GEMS project (Generic Eclipse Modelling System [152]) aims at bringing the GME metamodelling facilities to EMF in order to support rapid development of graphical editors. Please, note that it is still an *incubation* project. GEMS supports the graphical definition of a metamodel and generates a GEF-based [250] graphical editor for conforming models. Customization of the editor is based on a CSS style sheets mechanism. Currently, GEMS support basic importation of GME metamodels and models into GEMS, but reverse importation is still to be done.

To summarize, GME is an open-source framework that provides with all the needed capabilities to build M2DAT except from code generation. It is extensible, but in a quite challenging manner and its models cannot be exported to other tools, though ongoing work is focused on bridging GME and EMF.

2.2.9 Kermeta

Kermeta is metaprogramming environment that allows defining the structure and behaviour of (meta) models [175, 329]. It is based on an object-oriented DSL optimized for metamodel engineering and is fully integrated with Eclipse, including features such as an interpreter, a debugger, a prototype, an editor and various import/export transformations.

Its initial purpose was to enable metamodellers to give an operational semantics to their metamodels but it also works as a model transformation tool as we will describe in section 2.3.3.5.

A metamodel is defined textually in the Kermeta language. From that specification, you can generate a Kermeta model and edit it with a typical EMF reflexive editor or you can translate it to en Ecore model and use all the graphical capabilities of EMF to edit it. Therefore, since Kermeta metamodels are directly imported/exported from/to Ecore metamodels, Kermeta can be used as a roundtrip textual editor for Ecore models. In addition, you can specify the semantics of the model using the Kermeta language, a DSL that directly maps to the behaviour model. Moreover, it is possible to transform a Kermeta model, which contains semantic information into an Ecore model. The semantic is preserved within Ecore annotations. Once you have defined the dynamic semantics of the metamodel using Kermeta, you can execute any conforming model.

This approach is interesting because it contains a model-based representation of semantic information. Yet, it is not possible to create a customizable textual representation for the model itself. The main reason to include Kermeta in this discussion is because it allows to describe the semantics of a modelling language following a model-driven approach.

All this given, we may conclude that Kermeta is to be used to enhance the capabilities of EMF as a metamodelling framework by adding semantics to defined models. That is, it completes better that replaces EMF. Besides, the Kermeta language can be used for model-to-model transformations and to check constraints on EMF models, though it does not support model-to-text transformations. No editor (apart from those provided by EMF) is provided for terminal models. Finally, running atop of EMF lent it a standardized character and eases the task of extending and using it from other tools.

2.2.10 MetaEdit+

MetaEdit+ [243] is an environment for creating and using DSLs. It was initially conceived as a research prototype [187, 323] developed by the metaphor research group that later became a commercial product.

It provides a metamodelling language called GOPPRR attending to its components: Graphs, Objects, Ports, Properties, Relations and Roles, but no graphical editor for it. Therefore, metamodels has to be defined using a forms based interface. Although the concrete syntax is not explicitly separated from the abstract syntax, the use of different editors for each eases the distinction. For instance, a symbols editor allows connecting a symbol with each metamodel element. In addition, new symbols can be defined using a drawing panel. Once the metamodel has been defined, MetaEdit+ generates a graphical editor for conforming models.

Regarding model validation, additional constraints can be added to the metamodel by defining reports in a proprietary language of limited expressiveness. The very same language is used for code generation, thus the code generation capabilities are also limited. You can extend them invoking external routines coded with a GPL, but then you have to translate the metamodel to the GPL.

To summarize, MetaEdit+ is a robust and contrasted DSL framework. Apart from being commercial, its main drawback is that it is an isolated framework (DSLs will work just in the MetaEdit+ generated environment) without any support for model transformation and questioned code generation capabilities. Though some works have been done in both directions [191], they are still too incipient.

2.2.11 MOFLON

MOFLON is a metamodelling framework that supports also graph-based transformations [16] by adapting FUJABA [68] to work with MOF metamodels. Its metamodelling language is MOF and model validation is supported by defining OCL restrictions over metamodels. For model transformations, triple graph grammar rules are translated to JAVA code and QVT-compliance is planned and partially achieved. After you define a metamodel, MOFLON generates a JMI (Java Metadata Interface, [338]) API to handle conforming models.

Some tests with MOFLON have shown that, when compared with other metamodelling frameworks, it owns a low level of automation. Although it generates code from the metamodel specification, it is a set of JAVA disconnected packages. The user has to carry out the integration and configuration tasks in order to use the generated code. That is, instantiating a model after defining your metamodel is not a trivial task. Since Eclipse integration is planned for the next release (April, 2009) we hope it will help to solve this kind of drawbacks.

All this given, MOFLON provides with metamodelling and terminal models editing capabilities. In addition, those models can be transformed but they cannot be serialized into code. It owns a standardized nature that rests in the use of JMI and partial QVT conformance. However, since the standards chosen are not widely adopted nowadays, interoperability can be put into question. A movement towards EMF will help on this matter.

2.2.12 MOMENT

MOMENT is a formal framework for MOdel manageMENT [60, 61] embedded into the Eclipse platform that provides a set of generic operators to deal with EMF models. The underlying formalism is the algebraic language Maude [87]. MOMENT relies upon a set of generic operators to manipulate models and a set of bridges between EMF and Maude. The idea is to translate EMF models to algebraic specifications. The model management operators are Maude rewriting rules that works over such specifications. The results are translated back to the EMF technical space.

MOMENT does not provides metamodelling capabilities (though you can use EMF for this task) neither model-to-text transformation support. It implements partially QVT-Relations and supports the definition of OCL restrictions for model validation. Since it is an EMF component, we may qualify it as highly interoperable and extensible, though no information is available on how to extend it.

It is worthy mentioning that a new version was released when we were writing this dissertation (MOMENT2, November 2008) that improves the support for model transformations and model validation of MOMENT. As well, it enhances MOMENT’s aligment to standards.

2.2.13 openArchitectureWare

openArchitectureWare (oAW, [277, 369]) is a suite of Eclipse-based tools focused on code generation. We can look at it as an improved version of AndroMDA that takes the most of the advances of MDE, like the model handling facilities provided by EMF and the industrial-quality user interface of the Eclipse platform. In fact, all the oAW tools, like the textual editors, are Eclipse plug-ins that you can use separately.

oAW supports parsing of terminal models and supplies a family of languages to check and transform models as well as code generation from them. Although it is strongly connected with EMF, it can work with other models like UML2, XML or simple JavaBeans. Its use is based on the definition of workflows to specify generation/transformation executions. Besides, a number of prebuilt workflow components can be used for reading and instantiating models, checking them for constraint violations, transforming them into other models and then finally, for generating code.

This way, its main components are the Xpand [192] model–to-text and Xtend [123] model-to-model transformation languages; the oAW workflow engine and the Xtext [122] language for development of textual modelling frameworks, i.e. textual concrete syntaxes for DSLs. In addition, the Check language supports definition and checking of declarative constraints over a model (similar to OCL).

To sum up, we will not say that oAW supports metamodelling capabilities, since metamodels are actually Ecore models generated from a grammar. Model-to-model and model-to-text transformations are both supported, as well as model validation and terminal models edition (just with a textual editor). As it happens with Kermeta, being fully integrated in Eclipse and EMF results in high interoperability. Besides, it is extensible by nature.

2.2.14 TEF

The Textual Editing Framework (TEF, [311]) is an Eclipse plug-in for generating textual editors for DSLs. In turn, TEF generated editors are Eclipse plug-ins that provide with the traditional facilities of programming editors: syntax highlighting, content assist (code completion), intelligent navigation, or visualisation of occurrences. To that end, TEF provides with a language for defining textual concrete syntaxes in a set of templates. Each template describes the textual representation of a metamodel element. TEF is based on an abstract interface for modelling frameworks. This interface is implemented for EMF, but it could also easily be implemented for other technologies as well.

Therefore, a TEF editor is based on a metamodel and allows editing terminal models conforming to such metamodel. Thus, it does not support metamodelling since the metamodel is an input artefact from which TEF generates the textual editor. Besides no model transformation is supported. Althought it is built as an Eclipse plug-in, no documentation has been found on how to extend it.

Furthermore, since it is thought to complement EMF capabilities, it is fully interoperable, since TEF textual editors could be used along with any other EMF component. As well, being EMF-compliant make us qualify it as a standardized tool.

2.2.15 Whole Platform

The Whole Platform [326] is an Eclipse-based Language workbench for developing new languages. It provides with a textual metamodelling language. From there, you can define programs (terminal models) using the concepts included in the defined metamodel. To that end, the platform provides with a graphical editor that uses a set of generic notations already bundled (you may define new notations).

Since it is merely devised for metamodelling, it does not support model transformations, nor model validation. Note that it is based on Eclipse and GEF, but not on EMF. Thus, underlying storage formats are XML and JAVA what results on a medium interoperability level. However, it is worth mentioning it since seems to be rather powerful for defining new programming languages and building a minimum tooling for them.

2.2.16 XMF-Mosaic

XMF-Mosaic from Xactium is the last Eclipse-based metamodelling framework we mention. Its kernel is a MOF-based language called XCORE [85, 86] focused on the definition of executable languages.

In addition, it supplies languages for defining the tooling as well as the visual syntax (XTools), the textual syntax (XBNF), transformations (XMap) and restrictions (XOCL). The last is also used for semantics definition.

Its main concern is that it is a commercial tool, though a free evaluation version is available.

2.2.17 Others

In this section, we briefly introduce those tools or components less relevant from the point of view of this review due to its low adoption ratio, non-availbale information, late appearance or simply because of they have been already deprecated.

- **IEME**

The Integrated Eclipse Modelling Environment (IEME, [2, 3]) is a modelling environment based on Eclipse that aims at integrating existing Eclipse plug-ins to support MDE proposals. Therefore, the architecture of IEME is the closer to M2DAT from all previous works.

However, IEME seems to limit its contribution to define an architecture (close to that from M2DAT) of Eclipse plug-ins, but nothing is say on how they are integrated and more important, how they are to be used. Regarding technical issues, IEME does not support the definition of *customizable* model transformations. Besides, the validation support is limited to graphical models whereas as we will state on this thesis, the cornerstone of a MDE process is not the visual representation of the model, but the model itself.

Nevertheless, the main point with IEME is the absence of information apart from the referenced publications. Nothing has been found on the Web, nor the digital libraries visited (IEEE, ACM, Springer, and Elsevier) about IEME.

- **MDWorkbench**

MDWorkbench is an Eclipse-based IDE for code generation and model transformation [324]. It is said also to be a metamodelling framework supporting Ecore, UML and KM3 [181] metametamodels. Nevertheless, it is a commercial tool the free version is limited to work just with UML models.

It provides with transformation capabilities by means of a proprietary imperative language called MQL (Model Query Language). It owns a JAVA-like syntax and supports special operations to work with collections. In addition, ATL transformations can be used. Code generation is also supported by means of a template-based language called TGL (Text Generation Language).

Just as a matter of interest, it supports documentation of models in MS-Word format.

- **MOSKitt**

Modeling Software KIT (MOSKitt) is a free case tool, built on Eclipse and running atop of EMF which is being developed by the Valencian Regional Ministry of Infraestructure and Transport to support the gvMétrica methodology (adapting Métrica III [248] to its specific needs).

Regarding M2DAT, MOSKitt aims at supporting exactly the same capabilities using almost the same technologies (EMF, ATL, AMW, etc.). However, we cannot compare it with M2DAT since its first version was released in October 2008, when this thesis was almost finishing. Besides, so-far it just offers support for some common models, widely supported in Eclipse, like UML class diagrams or Business Process models.

Nevertheless, the advent of MOSKitt is another proof of the correctness of M2DAT proposal. It is a very similar tool that has been planned long after M2DAT was designed and that shares quite a lot of its technical design.

- **openMDX**

openMDX [278] is said to be a framework for MDA support. In fact, it seems to be just a J2EE code generation framework. It takes UML models conforming to MOF 1.4 and generates EJB, .NET or CORBA code. The generation process is based on the use of JMI [338]. It is worth noting that openMDX skips the PSM. It works directly with PIM models. The platform specific knowledge is encoded in the tool itself.

Therefore, we will not include openMDX in the section of model-to-text transformation languages since its primarily goal is broader than generating code from a model. In essence, it aims at providing with a complete information system from the source models. We might say that it encapsulates both model-to-model and model-to-text transformations in just one step since it skips the PSM.

- **PathMate**

PathMate is a commercial tool quite similar to openMDX. This time, code generation is limited to Java, C and C++ from UML models. Code generation is based on a proprietary template language, so called PathMATE™ Transformation Engine notation. However, code templates are customizable.

- **RoclET**

There are also tools dedicated just to provide with validation capabilities over terminal models. We bring here just one of them, RoclET [228].

It is an Eclipse plug-in that allows defining UML models and specifying OCL constraints over them. In addition, refactoring of constraint after refactoring the UML model is also supported. Nevertheless, it is limited to work with UML (1.5) models and, though OCL evaluation is supported, validation of models is still too immature.

Please note that there are similar proposals like OSLO, Octopus, etc. A good survey on this can be found at [71]. However, none of them fulfils our requirements for M2DAT. Thus, we include RoclET as an example since it is the closer to what we were looking for.

2.2.18 *Summary and Discussion*

In order to provide with an overview on existing proposals, Table 2-3 summarizes the main features of the works reviewed regarding the evaluation criteria described in section 2.2.1. The set of features considered are summarized on Table 2-2

Table 2-2. Evaluated Features on tools for MDE tasks

FEATURE	**DESCRIPTION**	**VALUES**
SCOPE	Commercial, Open-Source, Academic	C/O/A
METAMODELLING (MM)	Ability to define new metamodels	YES/NO
MODEL to MODEL (M2M)	Support for Model to Model transformations	YES/NO
MODEL to TEXT (M2T)	Support for Model to Text transformations: there is, there is not, limited	YES/NO/LMT
VALIDATION (VLDTN)	Support for models validation	YES/NO/LMT
GRAPHICAL EDITORS (EDTR)	Generation of graphical editors from the metamodel: Textual, Graphical, Not at all	T/G/NO
STANDARIZED (STDRD)	UML/MOF-Based or Proprietary Languages	YES/NO
EXTENSIBILITY (EXTNSBL)	Ease of adding new capabilities and/or modifying the already existing.	YES/NO/PRT
INTEROPERABILITY (INTRPRBL)	Ease of using functionalities provided by other tools.	YES/NO/PRT
EMF-BASED (EMF)	Whether runs on top of Eclipse EMF (or is planned)	YES/NO/TO-DO

Table 2-3. Frameworks and tools for MDE tasks

	SCOPE	MM	M2M	M2T	VLDTN	EDTR	STDRD	EXTNSBL	INTRPRBL	EMF
AndroMDA	(O)	NO	NO	YES	NO	NO	NO	YES	YES	TO-DO
ATOM^3	(O, A)	YES	YES	NO	YES	(G)	NO	NO	NO	NO
DOME	(O)	YES	NO	LMT	LMT	NO	NO	NO	NO	NO
DSL Tools	(C)	YES	NO	YES	NO	YES	NO	NO	NO	NO
EMP (EMF)	(O)	*YES*	*YES*	*YES*	*YES*	(G/T)	YES	YES	YES	YES
EMFATIC	(O)	YES	NO	NO	NO	(T)	YES	YES	YES	YES
GME	(O, A)	YES	YES	NO	YES	(G)	NO	PRT	NO	TO-DO
Kermeta	(O, A)	YES	YES	NO	YES	NO	YES	YES	YES	YES
MetaEdit+	GOPRR	YES	NO	YES	YES	(G)	NO	NO	NO	NO
MOFLON	(A)	YES	YES	NO	YES	NO	YES	PRT	PRT	NO
MOMENT	(O, A)	NO	YES	NO	YES	NO	YES	YES	YES	YES
OpenArchitectureWare	(O)	NO	YES	YES	YES	(T)	YES	YES	YES	YES
TEF	(O)	NO	NO	LMT	NO	(T)	YES	PRT	YES	YES
Whole Platform	(O, A)	YES	NO	YES	NO	(G)	NO	NO	NO	NO
XMF-Mosaic	(C)	YES	YES	YES	YES	YES	YES	NO	PRT	NO

The first and most important decision to take when developing a tool for MDSD is which metamodelling framework is going to be used. In some sense, this decision conditionates the rest of technological decisions.

In this sense, despite of the efficiency and performance of existing tools, none of them meets all the requirements for building M2DAT. ATOM[3] or GME lack of code generation capabilities. The DSL Tools or MetaEdit+ do not support model-to-model transformations. XMF-Mosaic is a commercial tool, what automatically discards it to build M2DAT. MOFLON owns a low interoperability and extensibility levels, etc.

That is, although there exist some all-in-all frameworks that aim at providing support for all the tasks related with model-driven development, hose frameworks do not fulfil our requirements to develop a new MDE tool. Specially, those related with tool interoperability. When using these kinds of frameworks, the result uses to be too tightened to the technology used. Hence, we argue in favour of combining the functionality provided by tools for specific MDE tasks to build your own tool. In other words, the only way of achieving full compliance with M2DAT needs is combining a set of tools that fulfil some of those needs in order to build a tool fulfilling all of them. Therefore, we opt for using EMF as underlying modelling framework.

EMF itself is an open framework, constantly evolving and integrating new projects. Any tool for MDSD built on top of EMF will be able to use the functionality provided by those projects. Therefore, using EMF we are ensuring that the new tool could integrate support for all the current MDE tasks already supported in the context of EMF (like model transformations, model validation, graphical editors development, etc.) but also for the new needs that might emerge as long as MDE advances keep growing. That is, using EMF we ensure rapid inclusion of emerging technology in M2DAT.

This way, if we need to support a new capability in M2DAT and there is an existing component providing it, we will be able to plug-in into M2DAT in an easy way. Even if there was no such component, we could develop it ourselves using the facilities provided by EMF. The integration with the rest of the tool would be effortless due to the extensible nature of the Eclipse platform.

Another key point is standardization. Although OMG standards have been the reference for MDE, a standard is useless without a reference implementation and EMF projects are not only the reference, but in some cases the unique implementation of some OMG standards or, at least, the most promising projects to implement the standard (like it happens with QVT).

As a conclusion we can say that there are very good frameworks and components for development of MDE tools. GME is a good example. ATOM[3] is another. However, the main drawback of these tools resides on the underlying framework. Since they are not based on a common platform, it is very hard to connect them with other existing tools or frameworks. In our opinion, *isolated* DSLs that do not shift information up and down the different abstraction levels are not helpful. For example, think of a DSL for component designing that you use to build your design models (deep down, your PSM). If it is a stand-alone tool, without any connection with the tool used to depict your analysis models, you are losing MDE promises of faster, less costly software development at a higher level of abstraction. In essence, the use of DSL frameworks comes out into a fully world of proprietary tools and languages for very specific purposes. This has not to be necessarily bad; the main problem is the absence of interoperability between them.

Finally, we would like to mention that when we started to work on this thesis EMF was just an emerging proposal. However, nowadays it has become the *de facto* standard for emerging technologies in the MDE field and the most succesful technical solutions for MDE are provided by EMF-based tools. Even existing frameworks are working on the development of EMF bridges (like the GEMS project for GME). Furthermore, the recent advent of new tools following the line proposal of M2DAT, like Moskitt (http://www.moskitt.org/) or Blueprint ME (http://www.atportunity.com/blueprintme.php) confirm that the bet for building an integrated MDE framework atop of EMF was correct. In other words, we would like to point out that, though nowadays the use of EMF as basis for building MDE tools is acknowledged as a common practice, it was far from being an obvious decision when we addressed the development of this thesis.

Finally, it seems EMF will keep its privilegiated status during the next years. In fact, forthcoming solutions to more recent problems are been developed in the context of EMF. For instance, one can think on metamodel evolution capabilities [84] and bridges between grammarware and modelware [380]. Actually, even those frameworks that existed before the advent of EMF, like GME, are driving their efforts to bridge the gap with EMF. Therefore, using EMF we ensure M2DAT a long-life of constantly improvement (because of the huge EMF community) and a lot of synergy with other exiting proposals (since more EMF-based tools appear each day).

2.3 Model-Transformation Languages

Although the study of transformation techniques has been a research topic for the last 30 years [69, 287], it was mainly focused on program transformations (source code). Model transformations had aroused little interest so far, but the boom of MDE and the advent of MDA have changed this situation drastically since model transformations play a key role in model-driven software development. As a consequence, a number of tools or langauges for model transformation development have arisen. Neverteheless, just by having a look at the different definitions for model transformation given in section 2.1.6, it becomes clear that model transformation is still an emerging research field.

These definitions, although being similar, leave some fuzzy points: the concept of what a model transformation is in essence (*automated processes*, a *program*, a *description*, an *algorithm*, a *model*, etc.); the level of automation that should support a model transformation proposal; the cardinality of the input and output models, etc.

If there is not even a complete consensus about what model transformation stands for, think on the complexity associated to choose one among the wide variety of existing proposals. To add complexity to this task, existing proposals could be classified according to a wide set of criteria. For instance, the number of input/output models, the approach they follow, the support for a graphical notation, the quality and quantity of documentation, the usability level, etc.

This task needs from a thorough study of the different proposals and, as we will show in this document, this was one the initial objectives this work. This section is devoted to present the main results on this matter. It is structured as follows: section 2.3.1 reviews previous works focused on classifying model transformation approaches. Sections 2.3.2.1 and 2.3.4 reviews the existing model-to-model and model-to-text transformation languages according to our own evaluation criteria. Finally, section 2.3.5 summarizes the main conclusions.

2.3.1 Previous Works on Classifying Model Transformation proposals

Our first step when we addressed the task of developing model transformations in the context of M2DAT was to get a complete understanding on existing languages. To that end, we started by reviewing all the previous works focused on classifying model transformation proposals, though there were not too many since model transformation was still emerging as a research topic. Even some of them aimed to present a new proposal, so they just included a classification aside but not as the main contribution of the work. This section summarizes our conclusions from reviewing those works.

- In [320] Sendall & Kozaczynski focus on studying the desirable characteristics that a model transformation language should have. Besides, they include a brief classification of approaches to model transformation definition. The most valuable conclusion from this work is that, according to the authors, the most recommended approach is some kind of **transformation language support** since the language can be adapted to the special needs of model transformation development. These languages, despite the different names used by different authors, are typically divided into *declaratives*, *imperatives* and the *hybrid* ones, that combine advantages from the both previous. At present, this approach is the most recognised: using a DSL for model transformation development.
- The work from Czarnecki & Hensel [97], which they revisited on [98], is probably the most referenced classification of model transformation approaches. Since the authors had focused their previous works in the study and definition of ontologies and feature models, their main contribution is defining **a taxonomy of model transformations** based on a feature model [96]. In pur opinion, despite the proposed feature model is complete and correct, it is too large and complex. A more simple and concise model would help on the election of a model transformation engine. In fact, the classification proposed is not even capable of defining a sub-category for each different value of the features identified.
- In [342] Tratt focuses on the maintenance of the traceability between the input and output artefacts as the way to reach real interoperability between modelling tools. After defining a set of simple steps to follow to develop a transformation engine that maintains the traceability information, Tratt presents a classification of techniques to define model transformations and concludes that the majority of the existing proposals follow a **declarative approach** since **results more suitable to support change propagation and traceability maintenance**.
- The last work we have considered explicitly to elaborate this state of the art can be found on the INRIA (*Institut National de Recherche en Informatique et en Automatique*) Web site [179]. Even though this work is the least formal, it results much more intuitive and **collects ideas spread all along the previous classifications**.

Since these classifications were made from the point of view of developers of model transformation approaches, they are too complex for non-experts in model transformation. We have to keep in mind that model transformations will be used by developers that have nothing to do with model transformation before. There are a vast amount of research groups that proposed model-driven methodologies, even before the MDE paradigm appeared. In fact, the traditional *requisites-analysis-design-implementation-testing* life cycle from the unified process [173] does not differ too much from the more thriving MDE approach. This way, a lot of work is being done in order to adapt works coming from these traditional frames to the MDE approach. In this context mappings between models that until now were carried out by hand, have to be automated (at least in some extent) using the newly model transformation approaches. So, developers behind those proposals, such as we ourselves, have to face the task of selecting and using one among all the existing model transformation approaches. Next section aims to help on this task.

2.3.2 Model Transformation Approaches

This section refines the ideas spread the above-mentioned classifications to state a clear and simple classification of the main approaches to model transformation. Later on, we will use this classification to identify the approach adopted by the model transformation languages reviewed.

- **Direct Model Manipulation**. It is based in the fact that, any given programming language aided by the use of APIs, can be used to define transformations between models. The JMI (Java Metadata Interface) specification is by far the most common example [338]. Using these APIs a new representation of a given model can be generated, what can be considered as a model transformation. On the one hand, this approach is quite simple, since the provided APIs are defined in general purpose languages like JAVA, so there is no previous learning. On the other hand, these languages were not intended for direct model manipulation. Therefore, using them to define transformations in different contexts or implying models at different abstraction levels results too complex.
- **XML-Based**. This approach used to be related with the XML technical space [208] and the most typical situation is that in which the models are represented using the XMI (XML Metadata Interchange) standard and the transformations are defined using XSLT (XML extensible Style-sheets

Language Transformations). It suffers from the complexity and verbosity that entails the use of XSLT [342].

- **Template-based.** The code is embedded in code templates spread between programming directives in a similar way to JavaScript. Typical examples of this approach are the transformation mechanism found on ArcStyler [20], AndroMDA [17] and CodaGen Architect [87]. It is commonly related with model-to-text transformations and result too rygid for model-to-model transformations.
- **Graph-Based**. It combines graph theory [126] with typed graphs − graphs with attributed nodes [93]. Graph-based approaches gathers the advantages of a solid theoretical basis and the similarity between models and graphs. Therefore, some of the most recognised proposals (that will be reviewed later), like AGG [70], VIATRA [94] or AToM^3 [107] have adopted it presently. Though they are quite appealing from the formal point of view because of their mathematical basis, they do not result convenient for complex transformations. Due to its different nature from other approaches, Appendix B provides with a more detailed overview on this approach.
- The **Declarative** style (AKA Relational) is based on defining the relations that must be kept between the input and output artefacts. This way, if the defined relations are not satisfied, the appropriate modifications will be made over the output artefacts. QVT-Relations is the perfect example of a declarative model transformation proposal [273]. As previously mentioned, the declarative style eases the mainteinance of traceability links.
- The **Structure Driven** approach starts by creating the elements of the output model to later add the corresponding attributes and references. Later on, this approach has been referred as **imperative style** in the model transformation literature. In contrast with declarative languages, QVT-Operational Mappings exemplarizes imperative languages. Using an imperative language results much more intuitive since it is similar to GPL.
- Finally, **Hybrids** approaches combine the declarative and imperative styles. It is worth mentioning that the most recognised proposals follow the hybrid approach, advocating that the declarative style should prevail over the imperative one. So far, the most recognised languages adopt an hybrid approach, where the declarative style prevails.

2.3.2.1 Evaluation Criteria

The review of model transformation languages has been made from the point of view of deciding which language will be adopted to develop M2DAT transformations. Since there are a number of proposals, we have identified a set of features that will help us on classifying them to choose the one that best fits our needs. Those features are described in the following, next to the reasons for their election:

- **Scope.** [Values: Open-Source, Commerical, Academic]

  We want M2DAT to be open-source, thus just open-source model transformation engines will be considered. Therefore, we will identify whether it is a commercial tool or an open-source one and also whether it comes from academics.

- **Approach.** [Values: Declarative, Graph-Based, Hybrid, Imperative, Template]

  As we have presented in previous sections, model transformation languages may adopt a number of approaches: declarative, imperative, hybrid, graph-based (that are also declarative in essence), XML-based, etc. Indeed, even those languages that adopt a hybrid approach, bets for using a preferred programming style. Since some constructions are more or less feasible to code depending on the approach followed, we will identify the one chosen by each reviewed language.

- **Direction**. [Values: Unidirectional, Bidirectional]

  Bidirectional transformations are a mechanism for maintaining the consistency of two (or more) related sources of information [99]. In MDE contexts, they allow to compute and synchronize views of software models. They are a need for future improvement of MDE proposals if we want to cope with issues like metamodel evolution and model co-evolution or roun-trip engineering. Even, the QVT standard bets for a bidirectional transformation language. Therefore, we are quite interested in determining if the languages to review support bidirectional transformations.

- **Tooling**. [Values: Low, Medium, High]

  Since we aim at identyfing the best language to develop M2DAT model transformations, usability of the selected language will be a key factor to make a decision. In this sense, we are concerned about the quality of the toolkit associated with the language (if available). For instance, we would like to know if it includes an IDE with code completion, syntax highlighting and the like.

- **Documentation**. [Values: Low, Medium, High]

  Another key factor at the time of selecting the transformation language to use is available documentation. Actually, the novelty of those languages results in very few (if any) documentation. Since the developers of the language are focused on improving and evolving the engine, very little time is dedicated to document the language. We have confirmed so far that, when facing new technology, the most valuable information is users feedback. Therefore, when studying available documentation, we will not focus just on manuals, tutorials, how-to documents and the like. We are mainly interested in complete case studies of successful applications, newsgroups, wikis and any other collaborative environtment that promotes knowledge sharing.

- **QVT/MTL-Compliant**. [Values: None, Fully, Partially, Planned]

  Although QVT specification [273] was still to come when we started to work on this thesis, the RFP had been already publisehd [274]. We have already mentioned that we want to reach the higher level of standards compliance for M2DAT without compromising usability. This way, since there exists an standard for model transformations we should check how existing model-to-model transformation languages align with the standard. The same is valid for the MOF Model to Text standard [266], regarding model-to-text transformation languages.

- **Framework**. [Values: *name of the framework /* ---]

  As we have already mentioned when reviewing frameworks and components for development of MDE tools, some of them support their own transformation language. Thus, we must identify if each reviewed language is tightened to some framewok.

- **EMF-Compliant**. [Values: EMF, Non-EMF, Bridge available]

  Finally, if the language is defined to run atop of EMF, we get all the advantages derived from EMF in terms of interoperability and extensibility that we have already commented. Moreover, since we will use EMF to build M2DAT, we look for a language able to cope with EMF models without the need for an extra effort.

2.3.3 *Model-to-Model Transformation Languages*

In the context of MDE, it becomes obvious the need for a way to effectively define and apply the model transformations implied in any MDE process. Obviously, one can opt for using a GPL like JAVA or C# plus the EMF generated API for .Ecore models to code a model transformation. However, this would be a very tedious and challenging task whose development and maintenance cost does not make up for the benefits provided.

In response to this need, a vast amount of model transformations engines has been delivered during the last years. A set of the most contrasted, covering a wide range of the existing approaches to the problem can be found in [50, 64]. This way, we can find proposals based on the use of graph grammars [126], like [16, 70, 94, 107]; proposals focused on the definition of DSLs for model transformation [184, 211, 305]; CASE tool proprietary model transformation languages [20, 88]; or model transformation engines that work by translating the mapping rules to algebraic specifications expressed in formal languages [60, 222], etc.

In the following, we present some of them. Those that have been most commonly adopted and those that, though not so successful, own a special interest from the research point of view. For instance, this is the case of the different works focused on implementing the QVT standard. They are still quite immature yet interesting to be evaluated with a view to future standard compliance of M2DAT transformations.

Finally, note that this section will limit to describe the main features of existing languages. Later on, in Chapters 4 and 5 we will provide with more detailed descriptions of the selected technologies used to develop model transformations in M2DAT (section 4.4).

The following sections, provide with an overview of each selected model transformation language. As well, each section ends by highlighting the way they behave regarding the features listed above.

2.3.3.1 AGG

The Attributed Graph Grammars (AGG, [70]) system is a visual language to define graph-based model transformations. Its main feature is that both the source and target models will be labelled graphs owning attributes whose types could be primitive or user-defined types.

AGG may be used (implicitly in "code") as a general-purpose graph transformation engine in high-level JAVA applications employing graph transformation methods. Due to its rule-based character, AGG may also be near in the field of artificial intelligence. The AGG tool supplies graphical editors for graphs and rules plus a textual editor to add JAVA expressions.

Regarding validation in AGG, one can check the consistency of a particular graph by means of graph constraints. Besides, the consistency of a graph transformation specification can be checked by defining critical pair analysis to find conflicts between rules (that could lead to a non-deterministic result) and checking the termination criteria.

The Tiger (TransformatIon based Generation of modelling EnviRonments) project [53] uses AGG to generate GEF-based Eclipse editors from a formal, graph-transformation based visual language specification. It focuses on in-place transformations (endogenous transformations, in contrast with traditional source2target exogenous transformations) used, for instance, in refactoring, reconfiguration or runtime models of executable languages (transformations as virtual machines). Moreover, it supplies formal analysis e.g., it can check whether a transformation terminates and always produces the same output.

To sum up, AGG is an open-source, graph-based and uni-directional transformation language. It provides with a complete IDE to code model transformations and the home site offers quite a lot of documentation. However, there are no cases of successful application, neither newsgroups nor (active) user forums. It does not align with QVT and it is a stand-alone application that could be integrated with JAVA applications but with no available bridge to use EMF models.

2.3.3.2 ATLAS Transformation Language

ATL (ATLAS Transformation Language) [184] is a model transformation language framed in Eclipse. It supplies an IDE that incorporates facilities like dedicated editors, debuggers, code completion, syntax highlighting, metamodel registry, etc. It is based on the OCL specification [268] and it is mainly a declarative language, though some imperative constructions are allowed to ease the coding of complex transformations.

ATL is a component of the AMMA (Atlas Model Management Architecture) platform [48]. Other components of AMMA are the ATLAS Model Weaver (AMW, [114]), the KM3 metamodelling language [40] and the Textual Concrete Syntax language (TCS, [183]).

ATL transformations are always unidirectional. Source models are read-only, while target models are write-only. During the execution of a transformation source models may be navigated, but changes are not allowed, whereas target models cannot be navigated. The last version of ATL compiler (ATL 2006) provides with advanced capabilities, like multiples source patterns, rule inheritance, and endpoint rules .The language is very stable and mature and it is constantly improved. In addition, there is a huge amount of available documentation in the form of manuals, usage scenarios and user newsgroups.

The delay on closing the QVT specification plus the absence of a reference implementation has resulted in ATL been widely accepted as standard de-facto for model transformation development. Additionally, some works have been made to align ATL and QVT [182] and a QVT-Relations implementation based on ATL-VM [180] is on the way [155].

All things considered, ATL is a uni-directional, hybrid transformation language (declarative programming style is preferred) developed as an EMF component. It provides with a complete IDE and a wide range of documentation that covers manuals, newsgroups, metamodels and transformations zoos, etc. Although it is not QVT-Compliant, it will be aligned with QVT.

2.3.3.3 ATOM^3

In section 2.2.3, we have already presented ATOM^3 as a metamodelling framework with model transformation capabilities. In fact, it was developed focusing on defining a framework for graph-based model transformations.

ATOM^3 forces you to define your metamodels using its own metametamodelling language. This will limit the expressiveness of your metamodels. For instance, ATOM^3 metametamodel does not support composition associations. In general, this type of problems is common to all the languages that impose their own metametamodel. Some constructions you are used to employ when defining MOF (meta)models are just not supported by them.

ATOM^3 supports TGG and NAC (see Appendix B). However, although the framework bundles some templates, you should have Python programming skills (Python is the source language of ATOM^3) to define actions or pre- and post-conditions, which are usually needed when developing complex transformations. Likewise, ATOM^3 transformations are unidirectional and does not support explicit scheduling.

To sum up, we can state that ATOM^3 is a graph-based, unidirectional transformation language. It provides with an IDE to develop model transformations and complete manuals on ATOM^3 site. Nevertheless, very few applications are found. It is not aligned with QVT and there is no way os handling EMF models with ATOM^3.

2.3.3.4 GReAT

GReAT (Graph Rewriting and Transformation) [9] is the graph-based transformation language of GME (see section 2.2.8).

GReAT programs are typically executed using a virtual machine, called the GR Engine. It interprets the rewriting rules comprised in the GReAT program and supports debugging tasks. Once the transformation has been checked (and corrected if needed) it can be translated into C++ working-code (remember that C++ was the source language of GME).

Using GReAT, one describes the transformations as a sequence of graph rewriting rules that operate on the input models to build the output model. The rules specify complex rewriting operations in the form of a matching pattern and a pattern to be created as the result of the application of the rule. The rules (1) always operate in a context that is a specific sub-graph of the source model, and (2) are explicitly sequenced for efficient execution. They are specified with visual notation thanks to a graphical editor. Three languages constitute the core of GReAT:

- Pattern specification language. This language is used to express the construction that should be matched in the source model. It supports a notion of cardinality on each pattern vertex and each edge.
- Graph transformation language. It is a rewriting language that uses the pattern language described above. It collects the source model, destination model and temporary objects in a single model that has to conform to a unified metamodel. This way, only transformations that do conform to the metamodel are allowed. At the end of the transformation, the temporary objects are removed and the two models conform exactly to their respective metamodels. In addition, one can add guards to control the rule applications by means of boolean C++ expressions.
- Control flow language. It is the language to sequence GReAT rewriting rules. It supports a number of features.

In contrast with AGG or ATOM^3, GReAT supports explicitly scheduling. To that end, a data-flow graph states the order in which mapping rules are executed.

To conclude, GReAT is an open-source graph-based transformation language. It supports just unidirectional transformations and provides a complete IDE since it is integrated into GME. As well, it provides with a huge amount of documentation, including application case studies, forums, etc. It does not conform to QVT, nor plans to be. Finally, no way of using EMF models with GReAT is identified.

2.3.3.5 Kermeta

Kermeta was already introduced in section 2.2.9 when we presented the Kermeta framework. Kermeta language is an executable meta-language not specifically intended to model-to-model transformation. It was born as a refactoring of MTL [371], another language from the Triskell team

Kermeta follows the operational approach. It is similar to direct manipulation but offers more dedicated support for model transformation. A typical solution in this category is to extend the utilized metamodelling formalism with facilities for expressing computations. An example would be to extend a query language such as OCL with imperative constructs. Likewise, QVT-Operational Mappings follow the same approach.

Indeed, Kermeta takes the form of an object oriented imperative language (with roots in Eiffel and Java) for (meta)-model manipulation. Additionally to the imperative syntax, it provides OCL-like constructs and facilities to work with models. Finally, yet importantly, a simple aspect mechanism allows you to modularize your code and simplify the design of your transformation.

Its imperative approach results in a very different code from that of any language that follows the declarative paradigm. Instead of rules, Kermeta uses operations, which are basically very similar to operations or methods in object-oriented programming languages, such as Java.

It should also be noted that, although it supports Ecore as domain language (i.e. models expressed in Ecore can be imported to any Kermeta transformation) the input and output of metamodel and model data has to be taken care of by the programmer. Therefore, every Kermeta program has to load and save data explicitly by itself. In other EMF-based languages, the input and output metamodels and models can be specified outside the transformation code, using Eclipse Run Configurations.

Kermeta is available as Eclipse plug-in, providing with a debugger, syntax highlighting and in-line error detection. Each transformation is implemented by three Kermeta files. The first file implements the preconditions, the second file implements the operations and last file implements the post-conditions of the transformation.

Like any other imperative language. Kermeta is recommended just for relatively complex transformations, where the expressiveness of imperative constructions become essential. On the downside, features supported by other transformation languages, like the mentioned load of models, or automatic tracing support are completely missing.

Concluding this section, we can say that Kermeta is an imperative language devised for metamodelling purposes that is also used to develop unidirectional model transformations. Besides, it owns a huge quantity of documentation though the tooling is not as powerful as those from other reviewed works are. Finally, although it is not thought to work specifically with EMF models, exiting bridge allows using EMF models with Kermeta.

2.3.3.6 MOFLON-FUJABA

Section 2.2.11 presented MOFLON as a metamodelling framework with model transformation support [247]. In fact, its underlying model transformation is based on adapting FUJABA [68] to MOF and JMI code generation.

In FUJABA, graph rewriting rules are wrote using the SDM (Story Driven Modelling, [12]) language. We can look at SDM as a mixture of collaboration and activity diagrams. In fact, collaboration diagrams was the abstraction used to express graph rewriting rules in first releases of FUJABA. Those SDMs are translated into JAVA (JMI) code directly executable over JAVA (JMI) objects.

Like other tools, you can define rules scheduling in FUJABA by given them different priorities in order to resolve conflicts where more than one rule applies. This can improve the performance of a transformation engine.

Although MOFLON has not been widely accepted as a model transformation language, it has been commonly recognised as a good tool for models simulation.

To sum up, MOFLON is another graph-based transformation language that claims to be bidirectional. The toolkit for developing MOFLON transformations could be qualified as medium whereas available documentation is rather poor, especially regarding application examples. Finally, MOFLON claims to be QVT-compliant, since it implements the graphic syntax of QVT-Relations and no way of processing EMF models with MOFLON is supported.

2.3.3.7 MOLA

MOLA is another graph based transformation language [186]. MOLA source and target metamodels are expressed by means of UML class diagrams by using its own metamodelling environment, METAclipse. MOLA aims at combining graph rewriting rules with the structures of control of traditional structured programming languages. Each MOLA sentence is represented by means of a graph rewriting rule. Those rules are sequenced in the way of an activity diagram. That is, rules scheduling is supported in the form of control flow graphs.

Regarding evaluated features, MOLA is open-source and graph-based. In contrast with MOFLON, it does not support bidirectional transformations. The toolkit provided, though frugal is efficient. There is available documentation from its home site and it does not plan any alignment with QVT languages, nor with EMF.

2.3.3.8 RubyTL

RubyTL [307, 309] is a model transformation language embedded in Ruby [301], what influences its concrete syntax. It is a rule-based hybrid transformation language and includes significant features such as the organization of rules in phases [309]. Besides, if it is specified at the time of configuring the transformation execution, one might modify the source model.

RubyTL syntax is rather intuitive, though it is not based on OCL. This way, a RubyTL rule includes the following clauses:

- **from**, where the constructs of the source metaclasses are indicated;
- **to**, where the constructs of the target metaclasses are specified;
- **filter**, which holds a condition over the source constructs for the transformation to be enacted;
- **mapping**, which states binding relationships between source and target model constructs. A binding is a kind of assignment that indicates what needs to be transformed into what, instead of how the transformation must be performed.

An interesting feature of RubyTL is its transactional behaviour. If some errors arise during the execution of the transformation, the target model is not created.

To outline the main features of RubyTL, we can say that it is an open-source hybrid transformation language (declarative style is preferred) that supports also code generation through a DSL plus code templates. It provides an Eclipse-based IDE, called AGE, that includes a Ruby editor with syntax highlighting, code templates and some code completion. Currently, one of its main drawbacks is the lack of available documentation and successful use cases showing its application. Likewise, it does not plan any QVT alignment. Though it was not developed to run on top of EMF, it works efficiently with Ecore models without raising any problem when they are imported.

2.3.3.9 Tefkat

Tefkat [211, 212] is a declarative, logic-based transformation language defined in terms of a MOF metamodel. It was initially developed as a response to the OMG's QVT RFP [274]. It supports single-direction transformation specifications from one or more source models to one or more target models. The transformation specifications are constructive, meaning that they specify the construction of the target model(s). There is currently no support for in-place update of models.

The Tefkat implementation is based on EMF and supports transforming native Ecore models as well as those based on MOF2, UML2, and XML Schema. It is usable in both standalone form and as an Eclipse plug-in with a source-level debugger.

In contrast with OCL-like syntax adopted by other languages, Tefkat's is similar to SQL and it results specifically designed for writing scalable transformations using high-level domain concepts rather than operating directly on the XML syntax. However, the concrete syntax is decoupled from the abstract syntax (the transformation model). Thus, Tefkat engine can be adapted to import model transformation specifications defined in different languages.

Tefkat supports templates and pattern definitions to encapsulate and reuse common expressions. It has a good support and tutorials. In addition, a Tefkat–Fujaba's TGG bridge was presented in [163].

In summary, Tefkat is an open-source language that follows the declarative style to support unidirectional transformations. As other EMF-based languages, it extends the Eclipse GUI to provide with an efficient IDE. Likewise, it lacks of application case studies and complete reference manuals. Finally, being an EMF component, it is fully functional to work with EMF models.

2.3.3.10 VIATRA

VIATRA [28] is another graph-based language that runs on top of Eclipse (not EMF) that uses its own metamodelling language, based on algebraic specifications: VPM [359]. It has served as the underlying model transformation technology of several ongoing European projects mainly in the field of dependable systems.

Mapping rules in VIATRA are expressed by means of graph rewriting rules that capture elementary transformation steps. They are combined using abstract state machines (ASM, [164]) to build complex transformations. Those state machines provide a set of common control structures with precise semantics frequently used in imperative or functional languages. This way, the ASMs act as control structures to reduce non-determinism and improve run-time performance. This identifies VIATRA2 as a hybrid language, since the transformation rule language is declarative but the rules cannot be executed without an execution strategy specified in an imperative manner.

The language used to implement all these concepts is the VIATRA Textual Command Language (VTCL). This language is primarily textual, thus VIATRA does not support graphical definition of model transformations presently.

Although it provides with exporters/importers, we can confirm that they do not work as expected with complex metamodels.

VIATRA2 (the last release) provides support for generic and meta-transformations [360] that allow type parameters and manipulate transformations as ordinary models, respectively. This allows arranging common graph algorithms (e.g. transitive closure, graph traversals, etc.) into a reusable library, which is called by assigning concrete types to type parameters in the generic rules. Furthermore, transformations can be externalized by compiling transformations into native Java code, as stand-alone transformation plug-ins. VIATRA2 transformations may call external Java methods if necessary to integrate external tools into a single tool chain.

As stated, one of the main differences with AGG or ATOM^3 is the support for explicit scheduling by defining abstract state machines to schedule the execution of the mapping rules.

Our main concern with VIATRA is that it is tied to its own metamodelling language. Though importers/exporters are provided, we can state that they do not work as expected with complex metamodels.

In addition, though there is available documentation, the syntax is not very intuitive. Besides, there is no way of defining auxiliary functions in VIATRA. The whole transformation must be coded inside the rules.

Moreover, when we have tested VIATRA we have encountered serious problems at the time of importing the UML-Ecore metamodel and conforming models. As well, target models are defined in the VIATRA format that raises some problems when imported in EMF. To sum up, though VIATRA provides with EMF-bridges, it does not work properly for every scenario.

2.3.3.11 QVT

Since the OMG released a standard for model transformations, we believe that special attention must be paid on it. After a minor introduction, next subsections will give a brief overview on the available implementation at the time of writing this dissertation. However, we would like to mention that, so far, there is little or no agreement at all on a reference implementation for QVT. The absence of such a reference implementation has definitively acted against QVT adoption. Moreover, existing attempts have shown that the standard still presents some fuzzy points.

QVT Overview

The Query/View/Transformations standard (QVT, [273]) is a family of languages for defining transformations. It defines two user-level languages, QVT-Operational Mappings and QVT-Relations, plus a low-level language that can be shown as the byte code of QVT, QVT-Core. Figure 2-13 shows an overview of the QVT architecture.

Besides, the mappings from QVT-Relations to QVT-Core is specified. This is a mapping of interest to possible implementers, but with no utility for mere users.

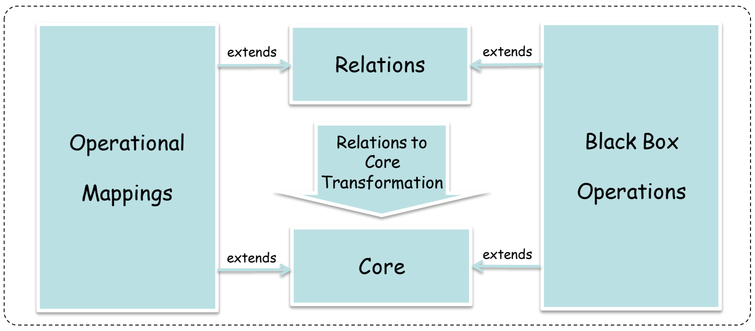


Figure 2-13. QVT Architecture

- QVT Core is a relational language (declarative) that supplies the set of basic constructions that allow defining source and target patterns and variables binding. QVT Core forms the basis for the other two languages and is not really meant to be directly used (as far as we understood it).
- QVT-Relations is another declarative language defined in top of QVT-Core. It supports complex expressions and a graphic notation.
- Finally, QVT-Operational Mappings is an imperative language that extends the previous.

In addition, black-box operations should be supported to allow calling external programs during transformation execution.

Until the final version of QVT specification was released, there were several projects focused on building a model transformation engine that fulfilled QVT RFP [274]. With the advent of the final adopted specification, many of them were abandoned whereas some new appeared. Though none of them has been able to provide with a complete implementation that include the three QVT languages so far, there do exist some promising works that are contributing to improve the specification.

In the following subsections, we enumerate the most relevant, distinguishing those that implement QVT-Relations from those that implement QVT-Operational Mappings. Please, note that we will not focus on studying how they behave regarding the features that compose the evaluation criteria, since we will not use any of them to develop model transformations in M2DAT because of two main reasons:

- They did not exist or was just incipient when we addressed M2DAT design and development.
- They are still quite immature due to the delay on providing a QVT final specification and the fact that it still presents some inconsistencies that constantly arise as long as QVT implementers progress.

In summary, since QVT tool support is still in its infancy [210], the following review on QVT implementations aims at identyfing the most promising works for future aligment of M2DAT transformations with QVT standard.

2.3.3.12 QVT-Relations Implementers

In the following we review the main projects focused on implementing the QVT-Relations language.

mediniQVT

mediniQVT [174] is a commercial product from ikv++ integrated in Eclipse that, up to now, seems the more stable and mature implementation of QVT-Relations. It is freely available under Eclipse Public License with non-commercial purposes.

mediniQVT includes tools for convenient development of transformations, such as an graphical debugger and an editor with code completion. It supports bidirectional transformations but suffers from some drawbacks when coding model transformations sketched on section 5.3.4.2. Besides, it works atop of EMF. Indeed, it is distributed as an Eclipse plug-in. Documentation is poor, almost reduced to the QVT-Relations specification itself, plus some notes on how to use the Eclipse GUI. There is a lack of real applications and the newsgroups are not very active.

ModelMorf

ModelMorf [343] is also a commercial implementation of QVT-Relations developed by TRDCC, a subsidiary of TATA Consulting Services. Though it is integrated on the Eclipse platform, it does not use EMF. Thus, models and metamodels are defined in terms of its own metamodeller.

Currently, not all features of QVT-Relational are supported in ModelMorf. Most notably, there is no support for incremental transformation execution, transformation extensibility and graphical syntax.

ModelMorf offers no development environment or graphical user interface. Transformation code may be created using any text editor and executed by calling the executable with parameters. Since there is no IDE, errors in the code are only detected when the transformation is executed, though useful messages are given in case of error. Application conditions are also supported by ModelMorf, via the when and where clauses of QVT rules. ModelMorf also builds up intermediate structures, for example for saving the tracing information.

Its main drawback is that, at present, it seems to be an abandoned project since the Web site has not been update since the version 3 of its beta was released (December 2006) with no clear sign of a forthcoming release. In addition, there is very few documentation and we have not been able to get a version of the engine.

Nevertheless, it is worth mentioning that Sridhar Reddy, that was a major influence on ModelMorf, has been also the greatest influence on the QVT-Relations specification chapters.

MOMENT-QVT

The MOMENT-QVT is a prototype integrated in the MOMENT framework (see section 2.2.12) that provides with partial implementation of QVT-Relations based on the term rewriting formalism MAUDE [87]. Since MOMENT works with EMF models, MOMENT-QVT allows defining transformation between EMF models.

Besides it provides with a QVT-Relations editor that supports syntax coloring, editing facilities and parsing facilities. Once the model transformation is defined by using the concrete syntax of the QVT Relations language, it is parsed to get a QVT model definition.

MOMENT-QVT provides support for traceability, in the sense that a traceability model definition, which records what objects of the target model definition have been generated from objects of the source model definition, is generated in an automated way during the execution of the transformation.

Declarative QVT: QVT-Relations in Eclipse M2M

The Eclipse M2M project (http://www.eclipse.org/m2m/) is a subproject of the Eclipse Modelling Project that provides a framework for model to model transformation languages. In particular, there are three transformation engines that are developed in the scope of this project: ATL (see section 2.3.3.2), Procedural QVT (Operational) and Declarative QVT (Relational and Core).

So, Declarative QVT is an Eclipse M2M subproject that aims at providing a implementation of QVT-Relations. In the M2M project proposal it was said that "An exemplary implementation will be for the QVT Core language, using EMF as implementation of Essential MOF and the OCL implementation from the OCL subproject. The main deliverable for this part of the project will be an execution engine that supports transformations. The engine will execute the Core language in either interpreted or compiled form. Following Core, the M2M project will provide an implementation of the QVT Relations language, based on the QVT Core execution engine, EMF and OCL. For both languages full language support will be delivered."

The QVT-Relations (QVTR) project was initially led by Compuware, who passed the baton to Obeo (industrial partner of AtlanMOD , the research group behind ATL) on July 2007.

Actually, the QVT-Relations implementation targets not QVT-Core but the ATL Virtual Machine (ATL VM, [180]). The implementation aimed to map QVT-Relations rules to ATL VM byte code, like QVT-Relations->QVT-Core mappings are described in the specification. Then, a transformation will be launchable by providing the ATL VM with a compiled version of the transformation, the models on which it must run and the metamodels to handle them. The first build of Declarative QVT was available in October 2008. Trace models, bidirectional and incremental transformations are not supported yet, but planned to be. A new version is to be bundled in the forthcoming release of Eclipse, Galileo.

Another Eclipse project, closely related with Declarative QVT is UMLX, led by Ed Willink [379]. Basically, it is a concrete graphical syntax to complement the OMG QVT, that was born as a graphical transformation language based on UML.

It provides accurate QVT models and validating editors for QVT-Relations and QVT-Core. Indeed, the evolution of the QVTR to QVT-Core transformation in the QVT specification can be attributed to the usage within ModelMorf and validation within UMLX. A text editor for (among others) QVT-Relations is available as part of UMLX, as well as .Ecore models for the ASTs of QVT-Relations and other languages.

However, it seems that UMLX is in troubles at present. Not long ago, the author claimed that his progress on a enhanced QVT-Relations graphical language is very slow since there are numerous issues with OCL and QVT that he has to work on.

2.3.3.13 QVT-Operational Mappings Implementers

Next, we review the main projects focused on implementing the QVT-Operational Mappings language.

SmartQVT

SmartQVT [141] is a open-source JAVA implementation of QVT-Operational Mappings built on top of EMF. It acts as a compiler in the sense that QVT-Code is compiled to Java source code. This is accomplished by a two-stage architecture:

- The QVT Parser converts QVT textual syntax into the corresponding representation in terms of the QVT metamodel, i.e. it builds an abstract syntax transformation model from the QVT code.
- The QVT Compiler translates the QVT model to a Java program. It uses EMF generated APIs for the source and target metamodels to execute the transformation.

This way, the SmartQVT compiler might be used in connection with other tools capable of producing a QVT model conforming to the QVT metamodel [273]. Additionally, serialized QVT transformations conforming to the QVT metamodel can be loaded and executed at runtime.

The first versions, based on a Python QVT parser, did not support error detection, but recent versions of SmartQVT do. Additionally, syntax highlighting is available. However, since SmartQVT compiles QVT code to Java code, no QVT-level debugger is available.

According to the QVT standard, tracing information in SmartQVT can be retrieved using one of three different resolving operations:

- *resolveone*. Looks for a target object created from a given source object.

- *invresolve*. Reverse resolve, looks for a source object created from a given target object.
- *resolveIn*. Looks for target objects created from a source object by a unique mapping operation.

Additionally, SmartQVT also supports late resolve, which behaves just like described above, with the exception that the resolving operation is performed at the end of the transformation.

Borland Together / QVTO

Borland Together [59] is an Eclipse-based commercial suite quite aligned with OMG standards. It provides with UML and BPMN editors, BPEL4WS translation, definition of OCL restrictions and, finally, one of the first implementations of QVT-Operational Mappings.

Moreover, Together supports model-to-text transformations using JET templates (we will introduce JET in section 2.3.4.4).

Procedural QVT: QVT-Operational Mappings in Eclipse M2M

In addition, Borland is contributing to the Eclipse QVTO M2M project that aims at providing with an EMF open-source implementation of QVT-Operational Mappings. In particular, a text editor, parser, and interpreter for QVT-Operational Mappings has been contributed by Borland to write .qvto files with some cool capabilities, like the support for hyperlinks (from usages to declarations).

Actually, Together uses the QVTO from the M2M project but has some commercial add-ons like debugger or code-completion. Although working with QVTO you are losing these advantages and using a more instable version, in exchange, you get access to new features that are not part of the Together release.

QVTo from OpenCanarias

Finally, OpenCanarias, have developed an open virtual-machine implementation of QVT-Operational Mappings [306]. To that end, they base on ATC [130], a low-level, imperative model transformation language built upon Eclipse and EMF. They aimed at supporting QVT, but from an indirect approach to avoid the cost of potential changes in the specification.

The idea is to inject QVTo specifications into QVTo models. Those models are transformed into ATC models (that can be showed as a byte code for model transformation) that are executed on the ATC virtual machine. This way, other transformation languages will be also executable on ATC-VM. Just as Declarative QVT is compiled to ATL byte code (see section 2.3.3.12 above)

It is worth mentioning that, during the development of OpenCanarias' QVTo, several discussions on the newsgroups resulted in the solution of several bugs in other Eclipse projects, like the OCL one.

In addition, OpenCanarias' team plans to support QVT Core as an intermediate step towards QVT-Relations. However, this project will be much more challenging since ATC is imperative, in contrast with the declarative nature of QVT-Relations.

2.3.3.14 QVT Implementers Summary

To summarize, Table 2-4 gathers the main features of existing QVT implementers.

Table 2-4. QVT Implementers

		SCOPE	FRAMWRK	ENGINE	EMF-Based
Declarative	mediniQVT	(C)	Eclipse	JVM	EMF
	ModelMorf	(C)	---	JVM	NON-EMF
	MOMENT-QVT	(O)	Eclipse	MAUDE	EMF
	Declarative QVT	(O)	Eclipse	ATL-VM	EMF
Imperative	SmartQVT	(A)	Eclipse	JVM	EMF
	Borland QVTO	(C)	Together	Borland QVT engine	EMF
	Procedural QVT	(O)	Eclipse	Borland QVT engine	EMF
	OpenCanarias QVTo	(C)	Eclipse	ATC-VM	EMF

Some remarks can be made:

- The most mature QVT engines at the moment of writing this dissertation comes from the industry. Therefore, some time is needed in order to get an open-source QVT engine.
- Most of the implementers run in Eclipse and use the EMF framework as model management platform.
- Finally, it is not clear which is the best option in order to provide with a QVT engine: some kind of virtual machine implementation (MOMENT, Declarative QVT and OpenCanarias QVTo) or a direct implementation (mediniQVT, ModelMorf, SmartQVT)

To conclude, as we have already mentioned, QVT implementations need more time to get realy useful and reliable to be used in real projects.

2.3.3.15 Others

Here we mention briefly some languages that, due to the state of the project, its novelty or its low adoption are less relevant but that has some interest, either as forerunner of present languages or because they own some interesting feature.

BOTL

The Bidirectional Object-oriented Transformation Language (BOTL, [62, 229]) is a tool for object-oriented model transformations. It works with models conforming to metamodels defined with a simple sublanguage of MOF. This way, a UML-based graphical notation is used for BOTL metamodels and model variables.

Graph-rewriting rules are specified using an UML-like notation. To that end, an ArgoUML extension could be used in the past. Since BOTL always transforms object models, the source model is a class model that can be mapped to UML or MOF meta class. BOTL is not a good option presently, due to the available documentation and the state of the project that seems to be abandoned. However, it owns a research interest since it supports bidirectional transformations, one of the main research fields around MDE in forthcoming years [99]

MTF

Model Transformation Framework (MTF) [170] is a set of tools developed by IBM that allows the implementation of transformations between EMF models. To that end, it provides a simple extensible rule language based on relations (text-based) called the Relation Definition Language (RDL).

MTF supports dynamic mode restriction [98], i.e. it allows marking any of the participating in/out-domains as read-only, restricting them to a particular execution of a transformation. Essentially, such restrictions define the execution direction. Besides, it supports Java code to select, match, or construct parts of the model(s), via extensions and custom constraints, which allow you to extend the MTF mapping definition language. Indeed, the RDL owns a JAVA look & feel.

This work is part of IBM's involvement in the QVT standardization. It was developed partly in order to prototype concepts that were to appear in the QVT standard. The intention was to implement more of the specification over time.

We have found a problem in MTF that is recurrent in transformation languages that claim to follow the declarative style. In the MTF documentation, the announced programming style is declarative. However, since each relation call explicitly the relation to apply to its contained elements, the transformation relations are very close to *invoked* rules. This way the relations are organised to follow the structure of the model and the structure of the whole transformation is more sensitive to the model structure. Working this way, the writing of transformation is less intuitive and more difficult for complex mappings.

Another weak point is the lack of documentation and examples especially on the way to use UML Profiles and to set properties value of stereotypes elements

Xtend

Xtend [123] is a DSL for extending metamodels, for example with custom properties for a metaclass. It is used for template modularization and also enables aspect-oriented templates. As of version 4.1, Xtend can also be used for functional style model-to-model transformations. However, only simple and limited transformations are possible, in comparison to ATL, Tefkat and other languages reviewed so far.

YATL

The Yet Another Transformation Language (YATL, [290]) is a transformation language developed within the Kent Modelling Framework (KMF). It is a hybrid language designed to express model transformations and to answer the QVT RFP [274].

It is described by an abstract syntax (a MOF meta-metamodel) and a textual concrete syntax (BNF). A transformation model in YATL is expressed as a set of transformation rules. A YATL transformation is unidirectional. The source and target models are defined using a MOF editor (e.g., Rational Rose or Poseidon) and KMF-Studio is used to generate Java implementations of the source and target models. The source model repository is populated using either Java hand-written code or GUI generated code provided by the modelling tool generated by KMF-Studio. The major part of the development of this tool was done before 2005 and it seems to be withdrawn at present.

2.3.4 Model-to-Text Transformation Languages

This section focuses on another sub-group of specific-task tools: code generators. In fact, code generators are also model transformation engines producing models with a very low abstraction level. Another feature of these models is that they are textually represented. However, the advent of textual editors for high level models implies that textual representation of models is not just proper of low level models.

In the following, we provide with a brief overview of the most adopted proposals focused on generating code from models, though some of them might offer support for other MDE tasks.

All the reviewed engines follow one of the model to code approaches identified by Czarnecki et al. in [97]: template based and visitor based.

Template-based generation is similar to Web dynamic programming, like JSP (Java Server Pages, [334]) or ASP (Active Server Pages, [244]) pages. Each template contains text blocks and control structures so that the latter combine the text blocks with the information gathered forrm the source model/s.

By contrast, in visitor based approaches the correspondent code is written to an output stream while the internal representation of the model is *visited*.

2.3.4.1 Acceleo

Acceleo [1] is an open-source code generator natively integrated in Eclipse. It supports template-based code generation for J2EE (Struts/Hibernate), Java, C#, Php and Python. Acceleo tooling is quite complete and provides many features for templates editing, such as syntax highligthing, meta-model and scripts based completion, real time error detection and real time preview. Unfortunately, documentation is not so good.

To implement complex operations, Acceleo supports a kind of black-box operations coded in JAVA that can be invoked from inside the templates, so-called *Services*. This allows for extensibility while keeping templates clean and easy-to-read.

Acceleo is based on two frameworks for model handling: EMF and MDR (NetBeans project), therefore it is fully EMF compliant

2.3.4.2 Acceleo/MTL

The Model To Text project (M2T, http://www.eclipse.org/modeling/m2t/) is an Eclipse project focused on the generation of textual artifacts from models. In the context of this project, the MTL subproject was started at mid of 2008 in response to the final version of the OMG standard for model-to-text transformations [266]. Its objective was to provide with an implementation of such specification.

Likewise, with the advent of the standard, people from Acceleo made a movement towards standard compliance and took the lead on the MTL project. As a result, the development of a new version of Acceleo, aligned with the OMG standard, was addressed and the MTL project was renamed to Acceleo. The first releases have been liberated in March 2009.

The development strategy is to stop having two similar projects in two different places: Eclipse MTL and Acceleo, and promote just the Eclipse one as the next generation of Acceleo. Actually, this is not the case: although they are similar, the syntax of the two Acceleo versions is different. The most recent one is entirely based on OCL.

At the moment, perfect stability is provided just in Acceleo original version since the Acceleo/MTL project is still under continuous development.

2.3.4.3 AndroMDA

We have already presented AndroMDA [17] in section 2.2.2. However, we mention it here since its primarily goal is code generation. Actually, its goal was to provide with a complete MDE framework but right now, it is just a code generator.

AndroMDA follows the template-based approach, though templates are known as *cartridges*. It bundles a number of cartridges for current development kits, like Axis, jBPM, Struts, JSF, Spring or Hibernate. In addition, new cartridges might be developed by extending a generic cartridge so-called *Meta*.

AndroMDA tooling is just adequate and provided documentation is more than enough. Besides, EMF compliance is planned, though not supported at the time of writing this dissertation.

2.3.4.4 Java Emitter Templates

JET (Java Emitter Templates) [293] is another component of the Eclipse M2T project that was developed by IBM. Indeed, it is the technology used for code generation in EMF, although a migration to an adapted version of Xpand (see section 2.3.4.6) is quite probable.

Its distinguising mark is its JAVA-like syntax what makes a JET file looking as a JSP page. Thereby, JET is especially appealing for JAVA developers. In contrast with some of the mentioned languages that use OCL or adapted OCL for navigating models, JET uses XPath [382], what results in too complex expressions when navigating source models. That is, the main problem of JET is that it is too JAVA-oriented. In fact, JET admits any XML file as input. It was not devised to work specifically with models.

2.3.4.5 MOFScript

MOFScript [262] was one of the first submissions in response to OMF’s RFP for a model-to-text standard [267] developed in the context of the European projects Modelplex and Modelware. It follows a visitor-based approach, is fully integrated in Eclipse and uses the EMF model handler. Actually, a MOFScript program is basically a parser that navigates the source model while generates an output stream that will be the code produced.

Until recently, it has been the most commonly adopted, mainly because it is quite easy to use (the imperative approach result more intuitive to non-experts on MDE) and it was one of the first works on this line. Besides, MOFScript tooling and the companion documentation have proved to be enough to develop model-to-text transformations in different contexts.

2.3.4.6 Xpand

Xpand [178] is a statically template-based language for model-to-text transformation integrated in OpenArchitectureWare (see section 2.2.13) and thus in Eclipse. Indeed, it is one of the components of the Eclipse M2T project.

Xpand itself has basic syntax but uses an underlying expression language and Xtend (see section 2.3.3.15) to provide powerful model-to-text (and even model-to-model) capabilities. A transformation template is defined for a specific metaclass and executed on all the objects of the source model that conforms to such class. Besides, transformations can be composed and inherited. The output of a transformation template is a concatenation of literal code and properties of the model element.

It also uses EMF as model handler and its syntax is OCL-like, though not pure OCL. It is comparable to any modern template engine, for instance Velocity or Smarty. An interesting detail about Xpand is that an adpated version of the language is used in the GMF generation process. In particular, the GMF project has made a movement towards standard-compliance by refactoring its version of Xpand by removing the use of Xtend in favour OCL and Procedural QVT (see section 2.3.3.13).

To conclude, it is worth mentioning that Xpand tooling is rather complete as well as Xpand documentation. Nevertheless, no aligment with the OMG standard is planned.

2.3.5 Summary & Discussion

In order to provide with an overview of the existing proposals, Table 2-6 summarizes the evaluation of selected features for the reviewed works. Besides, sections 2.3.5.1 and 2.3.5.2 put forward the main conclusions and ideas gathered from such review. The features evaluated, that were presented in section 2.3.2.1, are summarized on Table 2-5.

Table 2-5. Evaluated features for model transformation languages

FEATURE	DESCRIPTION	VALUES
SCOPE	Commercial, Open-Source, Academic	C/O/A
APPROACH	Adopted approach: declarative, imperative, hybrid (prevailing some style), graph-based, template-based	DEC/IMP/ HYB→(DEC/IMP) /GRAPH/TEMPLATE
DIRECTION	Unidirectional / Bidirectional	UNI/BI
TOOLING	Capabilities of related IDE:	LOW/MEDIUM/HIGH
DOCUMENTATION	Available documentation	LOW/MEDIUM/HIGH
QVT/MTL-COMPLIANT	Aligment degree with the corresponding standard (may be future work)	FULLY/NONE/ PART/PLAN
FRAMEWORK	Framework in which the language is integrated, if there is one	*Framework Name/* ---
EMF-BASED (EMF)	It is based on EMF or, ta least there is a bridge available (EMF) based / (BRIDGE) available / NON-EMF	EMF/BRIDGE/ NON-EMF

Table 2-6. Model Transformation Languages

	SCOPE	APPRCH	DRCTN	TOOLING	DOC	QVT / MTL-Compliant	FRAMWRK	EMF-Based	
AGG	(O, A)	GRAPH	UNI	HIGH	MEDIUM	NONE	---	NON-EMF	M2M
ATL	(O, A)	HYB→DEC	UNI	HIGH	HIGH	(PART/PLAN)	AMMA	EMF	
ATOM^3	(O, A)	GRAPH	UNI	HIGH	MEDIUM	NONE	ATOM^3	NON-EMF	
BOTL	(O, A)	GRAPH	BID	MEDIUM	LOW	PART	---	NON-EMF	
GREAT	(O, A)	GRAPH	UNI	HIGH	MEDIUM	NONE	GME	NON-EMF	
Kermeta	(O, A)	IMP	UNI	HIGH	HIGH	NONE	Kermeta	BRIDGE	
mediniQVT	(C)	DEC	BID	HIGH	LOW	FULLY	Eclipse	EMF	
ModelMorf	(C)	DEC	BID	LOW	LOW	PART	---	NON-EMF	
MOFLON	(O, A)	GRAPH	BID	MEDIUM	LOW	PART	MOFLON	NON-EMF	
MOLA	(O, A)	GRAPH	UNI	MEDIUM	HIGH	NONE	MOLA	NON-EMF	
MTF	(O)	DEC	BID	MEDIUM	LOW	PART	Eclipse	EMF	
RubyTL	(O, A)	HYB→DEC	UNI	HIGH	MEDIUM	NONE	EMF	BRIDGE	
TefKat	(O, A)	HYB→DEC	UNI	HIGH	MEDIUM	NO	Eclipse	EMF	
VIATRA	(O, A)	HYB→GRAPH	UNI	HIGH	MEDIUM	NO	Eclipse	BRIDGE	
XTend	(O)	TEMPLATE	UNI	MEDIUM	HIGH	NO	oAW	EMF	
YATL	(O, A)	HYBRID	UNI	LOW	LOW	NO	KMF	NON-EMF	
ACCELEO	(O)	TEMPLATE		HIGH	MEDIUM	NO	Eclipse	EMF	M2T
ACCELEO -MTL	(O)	TEMPLATE		HIGH	LOW	FULLY	Eclipse	EMF	
AndroMDA	(O)	TEMPLATE		MEDIUM	MEDIUM	NO	AndroMDA	PLAN	
JET	(O)	TEMPLATE		HIGH	HIGH	NO	Eclipse	NON-EMF	
MOFScript	(O, A)	VSTOR-BSD		HIGH	HIGH	NO	Eclipse	EMF	
Xpand	(O)	TEMPLATE		HIGH	HIGH	NO	OAW	EMF	

2.3.5.1 On Model-to-Model Transformation Languages

After choosing EMF as underlying modelling platform to build M2DAT, the selected transformation language has to be able to work with EMF models. In some sense, one might think that the bet on EMF reduces the scope of the search. However, if you take a look at Table 2-6 you notice that limiting our selves to EMF-compliant tools is not a restricting decision at all. In fact, the opposite decision would be much more restrictive since the most commonly adopted and mature engines are EMF-based tools.

Taking this into account, the main conclusion obtained from the review is the selection of ATL as the (preferred) technology to develop model-to-model transformations in M2DAT. Nevertheless, we will test other engines running atop of EMF in order to ensure that our decision was correct. In the following we put forward some remarks related to these decisions, though we will elaborate more on this in Chapter 4.

During the last years, we have worked extensively in the development of model transformations. This background let us state that the most suitable approach to address the development of model transformations is to use a DSL that follows the declarative style. However, the aid of some imperative constructions is needed to keep a *readable* transformation. In other words, although purely declarative programming is enough for any (model transformation) task, in some scenarios the imperative alternative brings simplicity to the transformation. Therefore, we bet for an hybrid transformation language that follows the declarative style. As well, we discard the graph-based approach because its usability for complex transformations can be put into question, though it is probably more appealing from a purely researcher point of view. Likewise, though the fact that graph-based transformations can be represented graphically is another advantage, it might be extended to any model transformation approach since the final QVT specification defines a graphical notation for model transformations.

As mere users of the transformation language, one of our main concerns was to check the available documentation. A common problem we have detected is that the toolsmiths and the practitioners of each transformation language are the same: only those that have developed the language use it. This results in the lack of application scenarios and made us infer that they are not devised to be used by non-expert users. Besides, the recurrent use of toy examples (like the well-known class to relational [46]) instead of real complex case studies make us wonder about the feasibility of using those languages in real projects. In this sense, ATL is the best option since the ATL project site (http://www.eclipse.org/m2m/atl/) provides with a complete set of successful applications in projects from different domains. Additionally, the ATL newsgroup provides with a constant and valuable feedback where the user might find the answers to problems that some other have faced before.

As well, though present QVT compliance is not mandatory for us, plans for future alignment are a must. In this sense, ATL seems to be also the most appealing. As we have mentioned, a QVT engine running atop of the ATL-VM is under development in the Eclipse M2M project. Therefore, ATL and QVT alignment will allow us to translate M2DAT model transformations to standard-compliant transformations without an extra effortwhen such engine is finished.

Last, but not least, the coupling of ATL with the ATLAS Model Weaver [113] will let us develop *customizable* transformations with almost no extra cost. As we will show this ie one of the main contributions of M2DAT regarding how model transformations are handled in existing tools supporting model-driven development of software.

All these factors have worked in favour of the selection of ATL as model transformation technology for M2DAT. Nevertheless, we will revisit thoroughly some of the conclusions sketched here in Chapter 4.

2.3.5.2 On Model-to-Text transformation Languages

First of all, we would like to put forward our bet for model-to-text transformation as the way towards code generation againts stand-alone parsers. We will explain this decission on section 4.6.1

Once stated that we will use a (EMF-based) model-to-text DSL for code generation, we finally decided to use MOFSript. Among the reviewed works, the most mature ones when we started to work on this thesis were MOFScript, AndroMDA and JET. However, AndroMDA and JET did not not work with EMF models, thought it was planned to do so. Actually, AndroMDA will not do it since the project’s leader, Mattias Bohlen, has withdrawn it. Regarding JET, it was not devised to work with models. It is more a template-based code generation language than a proper model-to-text transformation language. As a result, EMF projects, like GMF, are replacing JET as code generation technology in favour of an adapted version of XPand. These facts made us decide for MOFScript.

Besides, MOFScript was the most contrasted since it was one of the first submissions in response to the OMG RFP for a Model-to-Text standard [267]. It provides with a complete tooling, including syntax highlighting, code completion and the like and it is also the most complete regarding documentation.

Furthermore, its learning curve is lower since it follows the visitor-based approach (imperative style). This way, coding MOFScript transformations results easier since it is similar to traditional programming.

Nevertheless, the template-based approach seem to be gaining acceptance. as the OMG's MOFM2T standard confirms [391]. This wasy, template-based languages that appeared after we decided for MOFScript, like Xpand are becoming widely adopted. Therefore, we are currently testing Xpand with the model-to-text transformations bundled in the reference implementation of M2DAT (M2DAT-DB, see Chapter 5). Notice that this is one of the advantges of M2DAT specification. At any time we are able to replace the technology used for a specific MDE task for another technology supporting the same task, at very low cost.

2.4 Model-Driven Software Development Tools

The focus of this section is to evaluate how existing tools support (if they do) the functionality provided by M2DAT. By contrast, section 2.2 reviewed existing tools for supporting MDE tasks to identify the best option for building M2DAT, while section 2.3 was designed to study existing model transformation languages in order to select one to be used in M2DAT. That is, previous reviews aimed at defining the development framework for M2DAT while the following is focused on comparing M2DAT with tools devised for similar objectives.

In particular, this section focuses on reviewing existing tools that support model-driven development of software for two speficic domains: Web Information Systems and modern database (DB) schemas (in turn, we will distinguish those that support development of XML Schemas and those focused on ORDB schemas). We focus on these domains because:

- M2DAT is a technical solution for model.driven development of Web Information Systems.
- M2DAT-DB, the reference implementation for M2DAT, is a tool for model-driven development of modern DB schemas.

2.4.1 Evaluation Criteria

We need to have effective criteria to compare existent tools for model-driven development of software. To that end, we focus on identifying a series of common features that are interesting from the point of view of MDE, such as visual notation used, handling of model transformations, etc. Next, we describe those features.

- **Methodology.** [VALUES: *name of the methodlogy /* ---]

  One of the main inputs in order to evaluate the tool is studying the underlying methodology. To that end, regarding tools for model-driven development of WIS, since all of them are the result of implementing a given methodology, we will provide with a brief overview on the methodology supported by the tool. As well, we might assume that the underlying methodology for tools supporting model-driven development of relational DB schemas is the different works from Batini et al. around the mapping of ER to relational models [33]. On the other hand, just some of the tools for model-driven development of XML Schemas support a methodology, and just to some extent.

- **Paradigm**. [VALUES: Object-Oriented, Data-Centered, Structured, Semi-Structured, Service Oriented].

  Although we can not state that each tool follows just one software development paradigm (apart from the model-driven one), all of them some prevails some style over the rest. Here we focus on identyifing which one is the preferred in each case. This way, the most of tools for WIS development follow the Object-Oriented paradigm, while all the tools for model-driven development of XML Schema follow the semi-structured paradigm.

- **Scope/Target**. [VALUES: Commercial / Academic / Open-Source]

  Once again, we want to identify if it is a commercial, an open-source or an academic tool. Note that the same tool could fall in two categories.

- **Modelling Basis.** [VALUES: UML, Ecore, RDF, XML/ E/R]

  This feature has to be evaluated in order to assess how complex it would be to align the tool with MDE standards. For instance, here we will focus on identifying if the modelling languages supported by the tool are UML profiles, MOF-based languages or Ecore-based languages.

- **Modelling Notation**. [VALUES: UML-like, Nodes & Edges, Nested Boxes, UML Profile, Tree-like]

  This point is directly related with the previous one. While previous point refers to the abstract syntax of the modelling languages supported by the tool, i.e. the basis of the underlying metamodels, this point refers to the concrete syntax. That is, in case model editors are provided, which is the notation used.

- **Validation.** [VALUES: None to Excellent]

We want to identify if the tool supports model validation in the form of restrictions defined at metamodel level and later checked over terminal models.

- **Standardization**. [VALUES: None to Excellent]

  Here we aim at evaluating the level of compliance to standards. In contrast with the studies showed so far, here we consider not only OMG, but also standards from other organizations, like RDF [387]. Notice that this feature is directly related with the modelling basis of the tool. As we have mentioned, using UML, Ecore or MOF as modelling basis implies a higher level of standardization.

- **Abstraction layers covered.** [VALUES: CIM, PIM, PSM, PDM]

  Some tools provide support for the complete development process, from user requirements to final deployment, while other cover just a part of the development process. One way to identify the concrete support provided by each tool is to state which abstraction levels it covers (CIM, PIM, PSM).

- **Extensibility**. [VALUES: None to Excellent]

  This feature evaluates the ease of adding new capabilities or modify the existing ones, either ad-hoc or connecting with other tools. The evolving nature of MDE implies the need to provide with extension mechanisms in order to integrate the implementation of new advances in the existing tool.

- **Usability**. [VALUES: None to Excellent]

  We refer to the ease of using the tool. This implies studying if the tool owns a user-friendly front-end, the quantity and quality of available documentation, not only manuals but also collaborative media, like forums or wikis, etc. For instance, we would like to know if self-configuration of model transformations is supported or it is the user who has to configure the execution of the model transformations that have to be carried out during the development process.

- **Interoperability.** [VALUES: None to Excellent]

  We are interested in identifying if the models handled by the tools, can be exported / imported to / from other tools.

- **Code Generation**. [VALUES: None to Excellent]

  This feature evaluates the level of code generation supported. It might be able to generate just some skeleton of the working-code, or it might generate a fully functional artefact.

- **Deployment Platforms.** [VALUES: J2EE, .NET, DB Logical Model targetted]

  Another interesting feature to study is which are the platforms targeted by the tool. To put it another way, which are the technological platforms the tool is able of generating source code for: J2EE, .NET etc. Notice that, when referring to DB Schemas we focus just on the logical model targeted (XML, Relational, Object-Relational).

- **Model Transformations (MT)**. We have already mentioned a number of times that model transformations are the key of any MDE proposal since they are the only way to automate them to support MDE promises of fast, less costly software development. Therefore, we are very interesting in carefully reviewing how model transformations are handled in each of the tools under study. To that end, we identify a set of model transformation features. Obviously, they only apply if the tool do support some kind of model transformation, either model-to-model or model-to-text.

  - **DSL**. [VALUES: None to Excellent]

    Since some of these tools existed before the advent of MDE, the mappings between the supported models are hard-coded in the tool. This is a bad practice, not only from the point of view of MDE, where it would be inadmissible, but also from the point of view of traditional software engineering, since it violates the principles of abstraction and modularization. Therefore, we aim at identifying to what extent can be said that the mappings are coded with an external DSL for model transformation.

  - **Automation.** [VALUES: None to Excellent]

    We are interested in the level of automation of the model transformations supported. Some tools just provide with mappings that imply the need of manual refinement of target models after transformation execution.

  - **Customizable**. [VALUES: None to Excellent]

    In our opinion, a completely automatic process from requirement to final deployment is not only unfeasible, but also not recommendable. Design decisions have to be introduced to drive the development process. In a MDE context where the different steps of the development cycle should be automated by model transformations, the only way of introducing such design decisions is providing with a mechanism to parameterize such model transformations. Therefore, we want to identify if the tool support some degree of customization to drive the execution of model transformations.

– **Supported Types**. [VALUES: CIM2PIM, PIM2PiM, PIM2PSM, etc.]

Another feature that must be studied is the kind of transformations supported by the tool. Typically, you find vertical transformations from PIM to PSM mappings, but also CIM to PIM and horizontal transformations (PIM to PIM and PSM to PSM) should be implemented to support. In fact, business process models (defined at CIM level) are gaining acceptance each day as a first step in the development process. Therefore, we have to provide with tools that support such models and the mappings from them to the rest of models that compose the system.

– **Formalization**. [VALUES: None to Excellent]

Finally, we would like to check if the mapping rules implemented by each tool have been formally specified in some way, whether using the QVT standard or some formal language. Formalizing the mappings before implementing them, leads to detection of errors and inconsistencies in the early stages of software development and can help to increase the quality of the built models as well as the subsequent code generated from them. Likewise, the formalization of mappings simplifies its later implementation.

In the following sections, we study how existing tools for model-driven development of WIS behave in relation with the features just described.

2.4.2 Tools for Model-Driven Development of Web Information Systems

This section provides an overview of frameworks supporting proposals for model-driven development of Web Information Systems (WIS) development, so-called Model- Driven Web Engineering (MDWE, [197, 252]). Tools falling in this category are the result of implementing methodological proposals for WIS development that covered the traditional aspects related with WIS, like presentation layout, data persistence, business processes modelling or architecture designing. All of them share a common basis: they define a set of models that have to be specified along the different steps of the development process.

The following section follows the same structure of the previous reviews. They first give a wide overview of the tool under study. Next, they conclude by summarizing how it behaves regarding the features collected in the evaluation criteria.

2.4.2.1 ArgoUWE (MagicUWE)

ArgoUWE [195, 196] was the result of extending the open-source modelling tool, ArgoUML [297], with capabilities for modelling the content and navigation structures of Web applications [206] comprised in the UWE methodology (UML-Based Web Engineering, [198]). Later, new functionalities to model the business domain and behaviour of Web applications driven by the workflow were also added. Besides, support for checking OCL constraints over UWE models was also bundled.

The main problem of ArgoUML (and thus ArgoUWE) was the non-support for UML 2.0. However, ArgoUWE is not supported any more, since the authors of UWE have shifted the focus to MagicDraw [258], another modelling tool based on UML. This way, technological support for UWE is now distributed as a plug-in for MagicDraw, so-called MagicUWE [220].

Although Argo/MagicUWE has proven to be rather efficient for modelling Web applications, its main drawback from the point of view of MDE is the way transformations are handled. It provides with a set of predefined transformations (from content model to navigation model, from that to the presentation model, etc.) that, at best, are embedded in the plug-in code. This way, it is rather hard to incorporate on the tool any modification over the methodology. Moreover, automatically derived models have to be manually refined by the developer [205]. By contrast, this issue will be solved in M2DAT with the use of annotation models processed by customizable transformations.

In addition, UWE models are said to be UML profiled models and thus UML-Compliant. Actually, the ATL transformations included in [205] shows that UWE models conforms to a common metamodel defined in the KM3 language [184]. Thus, UWE models are not UML-Compliant, but MOF-Compliant.

Extending MagicUWE is allowed, though it has not been conceived to be extended. Indeed, there is no documentation on how it has been developed, neither on how it could be extended. Moreover, you have to use the extending capabilities of MagicDraw, a tool whose main objective, in contrast with Eclipse, was not providing with an open-source IDE composed of extensible frameworks.

Since both ArgoUWE and MagicUWE have been developed on top of well-known industrial environments, we could infer that they are user-friendly. Moreover, the documentation available at the UWE site (http://uwe.pst.ifi.lmu.de/) contributes to MagicUWE usability.

By the time of writing this dissertation, MagicUWE did not provide with code generation capabilities. UWE authors were developing the support to generate code for the Java Server Faces platform from UWE models. Besides, in [206], JAVA code generation is tackled but it is still to be integrated in MagicUWE.

UWE uses UML-profiled models, thus they should be easily exported to other tools. However, the well-known problems around XMI versioning, etc. put this into question. This way, in terms of interoperability, the use of a common underlying modelling framework like EMF brings more advantages than the use of pure UML models. Besides, UWE models are defined at CIM, PIM and PSM levels and the targeted platform (once the code generation is integrated) is JAVA. Likewise, the authors are working to integrate Service Orientation on the UWE methodology [291]. Thus, Service Orientation capabilities on the UWE tool will delay for a while.

Regarding constraints checking, it is hard to say how they are implemented in ArgoUWE. Indeed, according to [184] they are implemented by means of ATL queries. Although this approach works fine, it is not the most efficient. The checking process returns a Boolean value stating whether the model fulfils all the defined constraints or not. It does not distinguish which was the restriction violated, neither proposes a tentative or fix to solve the problem. Nevertheless, according to [199], constraints are defined with OCL but are hard-coded in ArgoUWE using JAVA.

Something similar happens for model transformations in UWE. Once again, all the mappings comprised in [184] are implemented with ATL. Nevertheless, in [199] is stated that not only ATL, but also QVT (just for specification tasks) and hard-coded JAVA rules are used. As far as we know, the real situation is that, though there are ATL transformations between some of UWE models (not all of them) they are still to be integrated into MagicUWE. At present, they are hard-coded in the tool. Besides, though there are wizards and launchers to invoke the execution of the transformations, the output models need manual refinement. In addition, the user has no option to drive the mapping process. Any customization is done by means of such manual refinement over the output models. Regarding the type of the transformations supported. There are PIM2PIM and PIM2PSM mappings already integrated. Moreover, CIM2PIM mappings are described in [184], but still to be integrated in MagicUWE. To conclude, we would like to point out that only those mappings specified with QVT can be considered as formalized and, as mentioned before, this is not the case of all the mappings comprised in UWE.

Finally, although UWE models defined with ArgoUWE could be formally validated using Hugo/RT [29], there is no way to validate UWE models from MagicUWE nowadays.

To sum up, we will focus just on evaluating ArgoUWE features. ArgoUWE follows the object-oriented paradigm. The modelling basis is UML since UWE models are UML profiles. Therefore, the modelling notation is UML-like. Constraints checking is supported but hard-coded in the tool. We may qualify it as rather standardized since it was based on UML. UWE models cover CIM, PIM and PSM levels and they are translated into J2EE applications. ArgoUWE was not devised as an open framework, thus it owns a low level of extensibility. By contrast, you might use the capabilities provided by other tools with ArgoUWE models, since they are expressed in XMI (once again, this just a theoretical statement since XMI has proven to be rather unsuccessful). Besides, being integrated into ArgoUML plus available documentation results in an acceptable level of usability. One of the main concerns with the tools is the way mode transformations are addressed. Some of them are hard-coded in the tool. Some others are specified with QVT but we guess that when it comes to implementation, they are also hard-coded in the tool. Finally, some other are implemented with ATL. They give some level of automation to the development process, but the models generated have to be manually refined. Regarind abstraction layers covered, just PIM2PIM and PSM2PSM mappings are covered, though CIM2PIM mappings are to be integrated. Recently, some work to support Service Orientation has being undertaken. Finally, formal validation of some UWE models was supported.

2.4.2.2 WebRatio

WebRatio [4] supports WebML [6], a language for expressing the structure of Web applications with a high-level description. It offers different models, together forming a website, namely, structure, derivation, composition, navigation and presentation models. Since the version 5.0, it is released as a set of Eclipse plug-ins [5]. Although it was devised in academics, presently it is a commercial tool distributed by WebModels, a spin-off created in 2001 from *Polytechnics of Milan*.

WebML was born as a language for modelling data-intensive Web applications [79] based on the E/R model [82]. Earlier versions of the supporting tool, WebRatio, supported WebML metamodels defined in the form of DTDs (Document Type Definition, [390]). Needless to say, DTD is not the best approach for modelling purposes [38] and when compared with MOF as a metamodelling language, there is no space for discussion. Moreover, working this way the advantages that MDE bring to Web Engineering, such as a common exchange format or powerful model transformations, are lost. Indeed, model transformations, in particular code generation to JAVA and JSPs, were coded using the XSLT language [389]. We have already mentioned its shortcommings when used for complex transformations [342]. Currently, XSLT is aided by ANT and Groovy technologies but still seems too archaic to support a MDE approach.

Recently, a MOF-based metamodel (indeed, Ecore-based) was proposed and implemented [310] to overcome these drawbacks. The metamodel was derived semi-automatically from the DTD. In addition, some work has been done to express WebML metamodels in the form of UML profiles [251].

Although WebRatio editors are built on top of GEF [250] and they use a UML-like notation, the lack of an underlying common metametamodel hampers interoperability with other tools and obviously implies a low level of standardization for the whole framework. Code generation, though archaic, is quite efficient and the targeting platform is JAVA. WebRatio does not support modelling the business domain, since the model that drives the development process is the data model (an E/R diagram). Thus, we can infer that the CIM layer, at best, is covered in a very limited way. To conclude, WebML has incorporated extensions for workflow-driven Web applications and Web Services, we may say that it owns (limited) SOA capabilities.

Finally, although WebML and WebRatio sites provides with documentation on how to apply WebML, the use of WebRatio is not so documented or, at best, not freely available, since training courses are sold through the Web site.

2.4.2.3 WebTE

WebTE [234] is an UML tool that supports the XMI standard. From the models of the Web application defined in WebSA [235] and OO-H [237], plus a UPT transformation model (a language for model transformation [236] based on the use of UML specifications serialized to JMI), WebTE generates an integration model that is transformed into working code using Velocity templates [18].

WebTE does not provide with a graphical interface to define such models. In fact, it is just a Web interface to upload the mentioned models and launch the transformations.

Regarding the evaluation criteria, WebTE’s follows the object-oriented paradigm. Its modelling basis is UML while no modelling notation can be identified since it does not provide with model editors. Besides, there is no support to check constraints over the models handled. It owns a high level of standardization since all the artefacts are expressed in XMI format and covers CIM, PIM and PSM levels. There is no way of extending the tool since it is a closed environtment and the lack of graphical editors (apart from the Web interface to load the models to process) and documentation results in a low level of usability whereas the use of XMI provides a high level of interoperability (as mentioned a numer of times, this is tru just in theory). Some code is generated for J2EE and .NET platforms. It uses a DSL for model transformation (so-called UPT) and provides with fully automatic transformations. However, they are just CIM2PIM and PIM2PSM mappings and does not support any way of customization. Finally, we can say that the mappings are somehow formal since they are expressed with a UML profile. It seems the project is in an idle state, thus no support for Service Orientation has been incorporated. Finally, there is no support for formal validation of WebTE models.

As part of a fuller discussion on WebTE, we can state some issues related with the tool. First and foremost, the lack of model editors is a serious drawback. Indeed, it recommends the use of an UML editor since all source and target models are provided in XMI format. We have already mentioned the inherent problems of XMI.

In addition, UPT expressiveness might be put into question. We have reviewed its metamodel as well as some examples, and we cannot state how it will perform with complex models. Besides, those transformations are somehow hard-coded in the components invoked from the Web interface and the user has no way to drive their execution.

2.4.2.4 OOWS Suite

The OOWS Suite [347] is the framework that implements the OOWS method [136]. The OOWS method is an extension to the OO-Method for Web applications development that adds two new models, namely, presentation and navigation, to capture the navigational and presentational aspects of Web Applications. In turn, the object-oriented software development method (OO-Method, [289]) is an automatic code generation method that produces the equivalent software product from a conceptual specification of the system.

OOWS is similar to UWE since a navigation model represents the navigational aspects of a Web application as views of classes from a class diagram (we might see this as a content model). In contrast, a dedicated presentation model for further abstraction of the user interface is not available and the presentation aspect is integrated with the navigation.

In essence, the OOWS suite is a framework for integrating the business logic collected in OO-Method models with a Web Interface produced from OOWS models. Therefore, the contribution of the OOWS Suite is basically the GMF-based editors for OOWS models, plus Xpand templates (see section 2.3.4.6) to generate the Web interface. Finally, integration between component objects (COM+, J2EE, etc) and the generated Web interface (PHP-based) is done by means of XML messages.

OOWS models use a UML-like notation based on custom metamodels. In fact, an ad-hoc extension to OMG's BPMN is used to model the business process. Such business process model is the source of a model transformation that is coded in QVT-Operational Mappings. It returns a PIM navigation model that has to be manually refined and later transformed into a PSM. Finally, the code generation lies over OlivaNOVA, the commercial tool that implements the OO-method [76].

However, the navigational model of OOWS is actually defined by specifying a set of views over the classes collected in the structural model from the OO-Method. To use it in the OOWS framework, a previous XSLT transformation has to translate the XML format used by OlivaNOVA to XML-Ecore format. The imported model should not be modified in OOWS Suite since OlivaNOVA relies in it to later generate the business logic.

Without considering technical details, our main concern with the OOWS method is that it emphasizes the use of conceptual models for generating both presentation and behavioural aspects. The use of conceptual models for presentation aspects can be put into question, but for navigation aspects, it is categorically erroneous. Instead, behavioural aspects have to be captured in the business process models that should drive the development process. Indeed, the authors state that following their approach "some minor details are still to be fixed directly in the final code".

Regarding model transformations, the PSM has been completely omitted. Indeed, in [347] the authors state that "Each transformation engine is composed of four elements that define its code generation strategy: (…) An Application Model (PSM) for each target platform (Java, .Net, ASP) that represents its technological aspects. The application model does not need to be modified by analysts because there is a clear relationship between Conceptual Model elements and Application Model elements. For this reason, it is hidden inside the transformation engine." (p. 9). Obviously, hard-coding transformations in supporting tools is not a good MDE practice. In addition, asserting that conceptual elements are univoquely mapped to deployment components is also too ambitious.

To sum up, the OOWS suite adopts the object-oriented approach and is based on two types of DSLs, thus two types of modelling basis are identified: Ecore metamodel and ad-hoc metamodels (based on OASIS, a language for specification of object-oriented systems [288]). Its editors use a UML-like notation and no constraint checking over terminal models is supported. Note that, since PIM models are directly translated into code, nor CIM neither PSM levels are covered. We might qualify the tool as partially extensible since it is partially based on EMF (open-source and highly extensible) and OlivaNOVA (a commercial tool not extensible). Usability is disminished by the absence of documentation and again it is partially interoperable (EMF versus OlivaNOVA). Code generation is supported for J2EE, .NET and COM technological platforms. Model transformations are hard-coded in the tool in the case of OlivaNOVA, while some XSLT transformations allow combining OlivaNOVA models with those from the EMF editors of OOWS suite. They are fully automatic but does not lend any space to customization. Since it moves from PIM models to working code, just PIM2Code mappings are supported. No Service Oriented functionality is planned and no formal validation or specification of models is supported.

2.4.2.5 HyperDE

HyperDE (Hypermedia Developing Environment, [260]) is a combination of a MVC (Model-View-Controller [148]) framework and a development environment for creating semantic Web prototype applications. It is based on the Object-Oriented Hypermedia Design Method (OOHDM, [314]), the first method that postulated separation of concerns for Web applications, and its successor, the Semantic Hypermedia Design Method (SHDM, [217]).

This way, from OOHDM models plus a user interface specification (views) and following the MVC pattern, HyperDE generates the Web application. Actually, SHDM models are used, thus the object model is derived from RDF descriptions [387] that provides with semantic descriptions of both data and metadata.

HyperDE inherits a distinguishing trait from SHDM: all depicted models conform to a common metamodel that collects all the abstractions used along the development process.

It is implemented as a modification of the *Ruby on Rails* framework [301] where the persistence layer (ActiveRecord) has been replaced by another one based on a RDF database. All HyperDE functions are accessed via Web interfaces. In addition, HyperDE also generates a Ruby-based API to manage both the model and SHDM's meta-model.

In summary, HyperDE does not supply visual editors. The level of standardization is quite low according to OMG, since the metamodel is an ad-hoc metamodel and no modelling framework is used. In contrast, the use of RDF models can be showed as a step towards interoperability. Besides, extending the tool is feasible but challenging, though the Web site provides with a huge amount of documentation. Like the OOWS suite, it omits the PSM level and goes directly from PIM to working-code. Regarding SOA capabilities, we may state that it is aligned in some sense with Service Orientation since it is focus on the semantic Web. Finally, no model transformation (as understood in the MDE context) is bundled in the tool.

The results of this review confirms that although HyperDE has been deployed using DSL techniques, it does not support a proper MDE development process.

2.4.2.6 Others

In the following we mention some works less relevant, either because they are part of abandoned projects or just because they do not align with MDE principles.

W2000

W2000 [31] is a methodology that extends the HDM methodology (Hypertext Design Model, [151]), a hypermedia and data-centric Web design approach, but it also adopts some features from UML to support the concept of business processes.

W2000 abstractions are collected in a MOF-based metamodel. Although the toolset is said to be integrated in Eclipse, it seems to be a set of disconnected components [32]. It includes a GEF-based editor, plus a MOF repository based on MDR/Net beans [258]. Some constraints are externally validated over W2000 models, while the rest were still to be supported. As well, some transformation rules were defined with AGG (see section 2.3.3.1) but they were not integrated in the tool.

All this given, we can state that W2000's tool support is quite instable and immature. Moreover, from reviewed works it seems that the authors abandoned the project. We found no available documentation or download site for the tool.

HERA

HERA-S [348] is the evolution of HERA [361], a method for developing adaptable and customizable Web Applications following a navigational structure that is defined semantically. In turn, HERA-S supports the design of navigation-oriented Web structures over Semantic Web data.

The idea is to wrap the data modelled in a domain model with a Web interface modelled in a presentation model. To that end, an application model is used as intermediate step. Each metamodel is represented in RDFS [388], while terminal models are expressed with RDF [387]. In addition, a user/platform profile, plus an adaption model, allow personalization of the presentation according to user preferences and browsing platform.

Unfortunately, the implementation of the above process is encoded in a set of XSLT transformations, while the edition of each model is supported by MS-Visio add-ins. While the premises of HERA-S are promising, more mature technical support is necessary in order to consider it a real framework for WIS development. In addition, we cannot state this is a MDE framework.

All this given, we do not include HERA in the final discussion neither in the summary displayed in Table 2-8.

2.4.3 Tools for (Model-Driven) Development of (Modern) DB Schemas

In order to provide with a reference implementation for M2DAT, we have chosen to implement its content module, M2DAT-DB (MIDAS MDA Tool for DataBases). Considered in an isolated way, M2DAT-DB constitutes a complete framework for model-driven development of modern database schemas that supports the generation of an Object-Relational DataBase (ORDB) schema or an XML Schema (XML Schema Definition, XSD) from a conceptual data model.

To confirm that M2DAT-DB improves existing technology in the field, this section aims at reviewing existing tools.

During the last years, the extended use of XML as preferred format for both data storage and exchange has resulted in the advent of a number of XML (and XML Schema) editors. In the following, we review the most recognised or accepted, plus those that has a special interest from the research point of view.

In general, all of them share a series of features, the most common being the support for different views of the XML Schema, like visual editors based on nested boxes and textual editors with syntax highlighting and the like.

On the other hand, defining the scope of our study, it is important to acknowledge that most Database Management Systems (DBMS) support the Object-Relational model, both commercial, such as Oracle [282], SQL Server

[243] or Informix Dynamic Server [169], and open-source, like PostgreSQL [287], MySQL [337], etc. By contrast, there are no frameworks that support modelling of pure ORDB schemas. That is why we enclose between brackets the terms “model-driven” and “modern” in the title of tis section (and the sections related).

Therefore, in contrast to the review on tools for MDWE, this study will limit to give a brief description of the tools that has a similar functionality. To that purpose, we will introduce the most recognised tools for development of relational DBs to help on the understanding of M2DAT-DB capabilities. In this sense, we would like to reference also some works focused on applying the MDA proposal to Data Warehouse development, like the ones from Klimavicius et al. [194] or the ones from Trujillo et al. [231]. However, they are not really in the scope of this thesis.

2.4.3.1 Altova XML

Altova XML [14] is one of the first tools for XML Schemas development, and probably the most-accepted so far.

It supplies different editors offering different views of the Schema. Among them, we find a visual editor based on nested boxes plus a (-n almost) plain text editor that, at the end, is the one preferred by developers.

The distinguishing trait of Altova with regard to other tools is that, in response to the impact of MDE and the boom experimented by software modelling, an UML-like editor has been added recently.

Notice that Altova XML pays no regard to the conceptual model and focuses just on the XSD definition. In other words, like most of the reviewed tools, Altova XML works just with the XSD model.

2.4.3.2 Oxygen XML Editor

Oxygen XML Editor [283] main objective is edition of XML documents. However, it also bundles XML Schema editing facilities, similar to those from Altova, though a UML-like editor is not supported in this case.

In addition, it provides with some capabilities for working with relational databases, but they are limited to exploring capabilities. Thus, it has not been considered as a proper tool for designing DB schemas.

2.4.3.3 Stylus Studio 2008

Stylus Studio 2008 [333] is similar to Altova and Oxygen.

In this case, its distinguishing mark is the addition of mechanisms to execute mappings between XML Schemas. Indeed, the tool support the graphical definition of the correspondences between two particular XML Schemas. From that, it generates the XSLT or XQuery code that implements the mapping (after an intermediate translation to JAVA).

2.4.3.4 hyperModel

Regarding XML Schemas development, hyperModel [391] is probably the most similar tool to M2DAT-DB. Indeed, it is also integrated in the Eclipse platform.

hyperModel is the tool supporting Carlson's proposal for modelling XML Schemas with extended UML [77, 78]. However, note that like the aforementioned tools, it goes directly to the XML Schema model, without considering a conceptual data model.

While Carlson advocates in favour of modelling the XML schema with extended UML, M2DAT-DB supports a higher abstraction level by allowing getting the XSD model from a conceptual data model. That is, the development process in M2DAT-DB starts from a PIM, that works as the classical domain model used in DB development process. Such model makes no reference at all to the deployment platform. Therefore, the very same model could be used as starting point to generate the DB model for any other logical model, whether it is OR, relational or whatever. Indeed, M2DAT-DB uses this model to generate both the XSD model or the ORDB model.

2.4.3.5 Rational Rose Data Modeler

Rational Rose [173] is a product line that resulted from the evolution of a CASE tool whose core functionality was to manage software models of a system under development. Originally, it was based on the use UML as modelling language. It has been traditionally recognized as the tool implementing the Unified Software Development Process [177] and the most adopted by software engineers during a number of years. Indeed, a common mistake has been identifying the use of UML with applying such process.

During the last years, its ratio of adoption has decreased with the advent of Eclipse and other open-source modelling tools.

The XSD plug-in of Rational Rose is able to generate an XML Schema file from a UML class diagram or inject an XML Schema into a UML class diagram. Therefore, no XSD model is supported. It moves directly from the conceptual data model to code and viceversa. This way, though it provides with a bidirectional UML-XML Schema bridge, it does not consider a proper XML model. Furthermore, the mapping process is predefined and hard-coded in the tool. There is no option to change or customize it

2.4.3.6 Enterprise Architect

Enterprise Architect [327] is a commercial tool quite similar to Rational Rose. It also encourages the use of UML, can be integrated in Eclipse and provides with an extractor from UML class diagrams to XML Schema.

Since it is the most recent of reviewed tools, it seems to be more aligned with MDE principles. For instance, it bundles customizable code generation templates. Indeed, Enterprise Architect is said to be a MDE framework.

It provides with automatic transformation from the conceptual data model to a stereotyped class model that acts as the XSD model. However, the XSD generation is a little bit tricky. Indeed, the code generation template uses the conceptual data model to generate the XSD code. If you want to customize the generation, then you have to spread XSD stereotypes over your UML classes in order to drive the code generation process. That is, you are not using a proper XSD model, but polluting your conceptual data model with platform specific information.

Among the reviewed works, Enterprise Achitect is probably the most similar to M2DAT-DB (regarding XML Schemas support). However, despite the fact that it is a commercial tool, we have to consider the strange way of generating the XSD. It is done directly from the conceptual data model, though some UML stereotypes might serve to mark the model. By contrast, M2DAT-DB allows generating a XSD model from the conceptual model whereas Enterprise Architect just lets moving formard form PIM to code or PSM to code. No PIM to PSM is supported. In addition, M2DAT-DB allows using a weaving model to annotate the conceptual model without polluting it. This way, the conceptual model is still valid to be used in other context, like the ORDB schema generation.

Regarding support for DB schema modelling, it advocates in favour of using a UML class diagram to represent the conceptual data model. From such model, a template-based transformation generates a logical data model represented in extended UML (which can be showed as a PSM). Finally, the logical model is serialized into code by means of a model-to-text transformation.

This way, the functionality provided by Enterprise Architect is the more similar to the one from M2DAT-DB among reviewed tools, since it uses UML for conceptual modelling plus model transformations specified wth a DSL. In addition, it allows adding new ad-hoc transformations.

2.4.3.7 ERwin

There are a number of applications, tools or frameworks for Database management (ERWin [92], Enterprise Architect [327], Oracle Designer [281], etc.). Although they include capabilities for designing the underlying DB schema, their main objective is supplying an IDE for all the tasks related with DB administration. Thus, when you want to use them as simple modellers they tend to be complex and not very intuitive. This fact matches up with a common drawback of existing MDE frameworks: the low usability level [318]. This becomes clear when one examines, for instance, the complexity associated to install the Oracle Designer repository in order to use the Oracle modelling tool.

Moreover, since they do not support the OR model, there is no sense in providing with a complete review on all of them. Hence, we will focus on ERwin to provide with some highlights about these kind of tools.

A number of those tools consider the possibility of starting from a conceptual data model (PIM) represented with the E/R notation [82] as a previous step towards a relational model [89] (PSM), following the ideas gathered in [33]. However, those are not pure E/R models, but adaptations polluted with logical details that ease the (automatic) mapping to the logical model. In this case, it is worth mentioning that the transformation is hard-coded in the tool, although some options to drive the mapping from the conceptual to the logical model are allowed. All of them are commercial tools, what is typically related with good usability levels and, in some sense, we might say that they are somehow aligned with standards since they use both the (adapted) E/R modelling language and Codd’s relational model.

2.4.3.8 Others

We can mention other tools, like **Liquid XML Studio**, **XMLFox** or the Visual Studio add-in for XML Schemas development: **Microsoft XML Schema Designer**.

As well, a particularly interesting editor for this dissertation, since it is integrated in the Eclipse platform is the **Eclipse XML Schema Editor**, a visual editor quite similar to the already presented. Indeed, it might own a more austere look and feel, though it is equally functional. Its interest resides in the fact that, in the near future, XML Schema models generated with M2DAT-DB will be editable with this editor without the need for any integration task.

Likewise, regarding ORDB schemas development, there are a number of frameworks that encapsulate the object to relational mapping. As a matter of fact, object-orientation has been the preferred programming paradigm during the last years, while relational databases have been the technology par excellence to endow with persistence the data handled by any given application. At present, the most popular are the Data Access Objects (DAO) of J2EE [335]. They offer an interface between the application and the DBMS and there are several implementations quite popular, like Hibernate [178] or JDBAccess (http://jdbaccess.com/).

2.4.4 *Summary & Discussion*

To provide with an overview of the reviewed works, Table 2-8 summarizes the main features of reviewed tools for model-driven development of WIS, while Table 2-9 does the same for tools focused on database schemas development. To that purpose, we use selected features collected in the evaluation criteria described in section 2.4.1. They are summarized in Table 2-7.

Next, the main conclussions gathered from these reviews are presented in two different sections.

Table 2-7. Evaluation criteria for tools supporting Model-Driven Software Development

BENCHMARK	DESCRIPTION	VALUES
METHODOLOGY	Underlying Methodology	*Methodology's Name*
PARADIGM	Preferred development paradigm: Data-Centered, Object-Oriented, Semi-Structured, Service Oriented	DATA-CENTER / OO / SEMI-STR / STRUCT/ SOA
SCOPE/TARGET	Commercial / Open-Source / Academic	C / O / A
MODELLING BASIS	Basis for abstract syntax	UML/Ecore/RDF/XML/E/R
MODELLING NOTATION	Basis for concrete syntax	UML-Like / E/R / Nodes&Edges/ …
VALIDATION	Support for constraint definition and checking and formal validation	(-) to (★★★★★) (1)
STANDARDIZATION	Level of conformance to standards	(-) to (★★★★★)(1)
ABSTRACTION LAYERS	Abstraction Layers supported	CIM, PIM, PSM
EXTENSIBILITY	Ease of adding new capabilities	(-) to (★★★★★)(1)
USABILITY	Ease of using / Documentation	(-) to (★★★★★)(1)
INTEROPERABILITY	Ease of import/export-ing models	(-) to (★★★★★)(1)
CODE GENERATION	Level of code generation supported	(-) to (★★★★★)(1)
TECHNOLOGICAL PLATFORMS	Targetted deployment platforms	J2EE/.NET/ ... DB Logical Model

BENCHMARK		DESCRIPTION	VALUES
MT	DSL	Mappings implemented with a DSL for MT	(-) to (★★★★★)(1)
	AUTOMATION	Level of automation	(-) to (★★★★★)(1)
	CUSTOMIZABLE	Customizable mappings	(-) to (★★★★★)(1)
	SUPPORTED TYPES	Types of transformation supported	CIM2PIM, PIM2PIM, PIM2PSM, PSM2PSM, etc.
	FORMALIZATION	Formalization of mappings	(-) to (★★★★★)(1)

(1) LEGEND (for weightable fields)

SYMBOL	VALUE
–	NONE
★	POOR
★★	FAIR
★★★	GOOD
★★★★	VERY GOOD
★★★★★	EXCELLENT

Table 2-8. Tools supportinng Model-Driven Development of Web Information Systems

	ArgoUWE	WebRatio	WebTE	OOWS Suite	HyperDE	M2DAT
METHODLOGY	UWE Methodology	WebML	WebSA / OO-H	OOWS	OOHDM / SHDM	MIDAS
PARADIGM	OO	DATA-CENTERED	OO	OO	OO	SOA
SCOPE/TARGET	(A)	(O), (A) → (C)	(A)	(A)	(A, O)	Open-Source/Academic
MODELLING BASIS	UML	(E/R extension) / DTD	UML	OASIS / Ecore	DSL / RDF	Ecore
MODELLING NOTATION	UML-like	UML-like	Not Supported	UML-like	UML-like	UML-like
RESTRICTIONS	⋆⋆⋆	○	○	○	○	⋆⋆⋆⋆
STANDARDIZATION	⋆⋆⋆⋆ (UML-compliant)	○	⋆⋆⋆⋆ (UML-compliant)	⋆⋆ (Ecore-compliant)	⋆⋆⋆ (RDF-Compliant)	⋆⋆⋆⋆ (UML / Ecore)-compliant
ABSTRACTION LAYERS COVERED	CIM, PIM, PSM	PIM, PSM	CIM, PIM, PSM	PIM	PIM	CIM - PIM – PSM - PDM
EXTENSIBILITY	⋆	○	○	⋆	⋆	⋆⋆⋆⋆⋆
USABILITY	⋆⋆⋆	⋆⋆⋆	○	⋆⋆	⋆⋆	⋆⋆⋆⋆⋆
INTEROPERABILITY	⋆⋆⋆	○	⋆⋆⋆	⋆⋆	⋆	⋆⋆⋆⋆⋆
CODE GENERATION	⋆⋆⋆⋆	⋆⋆⋆⋆⋆	⋆⋆⋆	⋆⋆⋆⋆	⋆⋆⋆⋆⋆	⋆⋆⋆⋆⋆
TECHNOLOGICAL PLATFORMS	J2EE	J2EE	J2EE, .NET	J2EE, .NET, COM+	Ruby on Rails	Oracle, .NET, PHP

		ArgoUWE	WebRatio	WebTE	OOWS Suite	HyperDE	M2DAT
MT	DSL	★★★	★	★	★★	NO → EMBEDED	★★★★★
	AUTOMATION	★★	★★★★	★★★★	★★★		★★★★★
	CUSTOMIZABLES	○	○	○	○		★★★★★
	SUPPORTED TYPES	*CIM2PIM*, PIM2PIM, PIM2PSM	PIM2PSM	CIM2PIM, PIM2PSM	PIM2Code		CIM2PIM, PIM2PIM, PIM2PSM, PSM2PSM
	FORMALIZATION	★★★	○	★★★	○		★★★★★

Table 2-9. Tools supporting (Model-Driven) development of (modern) Database Schemas

		Altova XML	Oxygen XML Editor	Stylus Studio	hyper-Model	Rational Rose	ERwin	Enterprise Architect		M2DAT-DB	
METHODOLOGY		–	–	–	Carlson's	–	Batini's	Codd's	–	MIDAS-DB	
PARADIGM		SEMI-STR	SEMI-STR	SEMI-STR	SEMI-STR	OO	STRUCT	OO	OO	OO	
SCOPE		(C)	(C)	(C)	(O)	(C)	(C)	(C)		(C)	
MODELLING BASIS		XML	XML	XML	XML	UML	E/R	UML	UML	Ecore	
MODELLING NOTATION		Nodes & Edges	Nodes & Edges	Nodes & Edges	Nested Boxes	UML notation	E/R	UML		Pure UML	
		UML-Alike					Relational Tables	UML Profile		UML-Alike	Tree Editor
RESTRICTIONS		–	–	–	–	–	***	–	–	*****	
STANDARDIZATION		*	*	*	**	****	**	**	***	****	
ABSTRACTION LAYERS		PSM	PSM	PSM	PSM	PIM, PSM	PIM, PSM	PIM, PSM	PIM, *PSM*	PIM, PSM, PDM	
EXTENSIBILITY		_	_	_	*	**	_	_	**	*****	
USABILITY		***	***	***	***	****	****	****	****	****	
INTEROPERABILITY		_	_	_	**	**	_	**	*****	*****	
CODE GENERATION		*****	*****	*****	*****	*****	*****	*****		*****	
TECHNOLOGICAL PLATFORMS		XML	XML	XML	XML	XML	Relational	Relatnl	XML	OR	XML
MT	DSL	_	_	_	_	_	_	****		*****	
	AUTOMATION	*****	*****	*****	*****	*****	*****	****		*****	
	CUSTOMIZABLES	_	_	_	_	_	****	_	***	*****	
	FORMALIZATION	_	_	_	_	_	_	_	_	*****	

2.4.4.1 On Tools supporting Model-Driven Development of Web Information Systems

This study has focused on how a number of MDE issues are addressed by existing tools for WIS development. Concluding this section, we can say that the most mature and stable one is WebRatio. However, WebRatio presents serious drawbacks when it is confronted with MDE principles. Indeed it does not adhere to any OMG standard (or Eclipse implementations, currently de facto standards), nor uses model transformations; neither encourages interoperability with other frameworks.

Besides, Argo/MagicUWE seems to be the most MDE-friendly, in the sense that it adheres to OMG standards by using extensively UML profiles and OCL restrictions plus a DSL for model transformations, though some drawbacks have been detected regarding how *customization* and formalization are handled.

Nevertheless, the state of Argo/MagicUWE serves to prove the instability of the tooling support for MDWE. Indeed, UWE is in the process of migrating its whole framework to the Eclipse platform and replacing UML profiles for MOF-Based DSLs (actually, EMF-based ones, like M2DAT).

One tentative reason for this landscape might be that those methodologies appeared before the advent (or the boom) of MDE. Thus, they opted for adapting their proposals to MDE principles. So, they addressed (or planned to do so) the building or adaptation of (existing) frameworks to support the new nature of their proposals, but the adaptation processes carried out so far are still immature. From a MDE point of view, there is still much work to do in order to align those frameworks with MDE guidelines.

For instance, one common drawback, clearly stated by Moreno and Vallecillo in [252], is located at how they handle model transformations. Some of them just hard-code the rules in the underlying tool while some other use XSLT style-sheets. This fact results in a gap between the design of the Web application and the final implementation. According to MDA principles, these rules should be defined at a more abstract level. Although some proposals have already tackled this task (see [199] for UWE, [234] for WebSA and [253] for WEI), these improvements have still to be integrated in the corresponding tools. By contrast, all the mappings of M2DAT will be implemented using a DSL for model transformation.

Besides, none of the reviewed frameworks offers support for customizable or parametrizable transformations, neither for applying formal techniques over the models handled by the framework. We plan to address these issues in M2DAT.

Indeed, we have already showed how it will be done in previous works. For instance, in [357] we used weaving models to support model transformations driven by user decisions in a MDWE process.

Another common problem is interoperability. In this sense, the use of weaving models to automate model migration is becoming widely accepted. In [356] we showed how to apply this approach in a real industrial environment. Such approach is been studied as a way to automate tools interoperability. As well, the fact that it will be developed entirely on EMF conferes advanced features to M2DAT in terms of interoperability. The studies about existing MDE technology that we have presented state that the most of them are built as Eclipse-EMF components. Thus, M2DAT models could be handled by any of those components without the need for building any bridge.

Among the reviewed tools, just ArgoUWE supports constraint checking over terminal models. However, they are hard-coded in the tool. By contrast, M2DAT will support checking constraints over terminal models defined in a separate way. Working this way, we are able to separate models edition from model validation. We can invoke validation just when needed, and we can modify the constraints to be checked at any time, without having to worry about the rest of the components of the tool. Likewise, the reviewed tools do not offer any support for applying formal techniques over the models handled by the framework. In this sense, [249] showed how M2DAT models were translated to MAUDE [87] to support the definition and formal verification of properties, as well as its validation.

As well, one of the main contributions of M2DAT regarding previous works is extensibility. M2DAT will be completely extensible since all the components used to build it are open-source components, specially devised to accept modifications, improvements and incorporate new functionalities. This way, M2DAT will allow for rapid inclussion of emerging technology benefits. An immediate consequence is the ability to target new technological platforms.

Adding support for a new platform will consist in defining a new platform model to abstract the targeting platform, weaving such platform model with already existing PSMs to obtain new PDMs (Platform Dependant Models, that is, models that map the concepts captured in PSMs to the abstractions supported by concrete technological platforms) and finally serializing them into code. Although this is not immediate, the process and the techniques to do it are well identified, thus it is a feasible task, both in time and manner, thanks to its extnesible nature and modularized architecture.

Last, but not least we would like to make a reference to Service Oriented capabilities. In response to the impact of Service Orientation [392] in recent years, some methodologies are updating their proposals to adequate them to services paradigm. However, so far just UWE has worked on this direction, though their initial results are still to be integrated into ArgoUWE (we guess it will be done in MagicUWE, the new version of the tool). By contrast, M2DAT follows the SOA paradigm from the beginning, since MIDAS is a complete Service Oriented methodology. First results on this line have already been presented [106, 353].

All this given, this study has highlighted a number of problems in the area of MDWE and lends strong support to the idea that M2DAT features will serve to fill some gaps detected. Specially, how model transformations are handled in existing proposals, poor interoperability and extensibility and their level of formalization. Besides, the study has highlightened a growing trend in MDWE proposals towards developing their tools as Eclipse plug-ins, or at least, upgrading or re-defining them to be “Eclipse compliant”, like UWE, WebML and the OOWS suite. Since M2DAT will be built atop of EMF from scratch, it will be ahead of exiting tools in this sense.

Finally, the reader should notice that even though MDE is a widely accepted approach, MDWE is still relatively new: all the tools listed in this section are academic proposals. We can conclude that the most outstanding challenge for the developers of MDWE tools is to take their tools from academic to industrial environments.

2.4.4.2 On Tools supporting Model-Driven Development of (Modern) Database Schemas

Acording to Berstein [380], in some cases, object-relational and XML technologies are the best choice to build a DB. However, we have confirmed that, so far, there is no tool supporting a complete model-driven process for ORDB or XML schemas development.

Regarding ORDBs, this fact is probably due to traditional DB has always followed the relational model as is. Thus, the frameworks for DB design have limited their selves to such model. In some cases (very few), they opted for extending the framework to support the modelling of object-relational constructions. However, instead of improvement, the effect was to bring additional complexity to the original framework.

On the other hand, regarding XML Schema development, the main problem so far is related with the fact the XML was not considered as a proper DB technology until recently. Thus, there is a lack of methodological proposals for

XML Schema development following the traditional principles for DB design. That is, we found no works considering a conceptual data model and a logical (XML) model.

In fact, just hyperModel, Rational Rose and Enterprise Architect might be said to be model-driven tools. However, hyperModel just ignores the PIM, going from a UML-sterotyped model to XML code, Rational ignores the PSM, going from a class diagram (conceptual model) to the XML code, and Enterprise Architect just supportd the movement from PIM to code or PSM to code. No PIM to PSM mapping is supported. The rest of works just focus on XML Schema editing without any support for a conceptual data model. Obviously they do not support the mapping from a conceptual data model to an XSD logical model.

Even existing tools for traditional DB design, like ERwin, that starts from conceptual data models (PIM) represented with adapted E/R notations, do not consider pure PIM models since they are polluted with logical details that ease the (automatic) mapping to the logical model.

By constrast, M2DAT-DB starts from a pure conceptual model represented by means of a class diagram, that is mapped to the selected logical model, XML or OR. The design decisions that drive the mapping process and help on mapping conceptual abstractions to technological components, are defined separately in an annotation model. This way, M2DAT-DB provides with full separation between conceptual and logical models while preserving the ability to generate different logical models from the very same conceptual data model.

Besides, M2DAT-DB is the result of a continuous improvement and refinement of a methodological proposal for XSD and ORDB modelling developed during the last years. Hence, methods and technologies supported have been widely studied and validated before its implementation, one of the main concerns of MDE tools nowadays [160].

Once again, a common issue to the reviewed tools is the way they handle model transformations. Just Enterprise Architect uses a DSL. It is a template based language used both for model-to-model and model-to-code transformations. However, we can state that the language is only suitable for quite simple and direct transformations. The constructions supported are not enough to support complex transformations. The rest of tools that support some kind of transformation just hard-code it in the tool itself.

By contrast, M2DAT framework emphasizes the role of model transformations in MDE development and this is immediately captured in the mappings bundled in M2DAT-DB. All of them are formalized and later coded using DSLs for model transformation. In addition, the user can drive the execution of the mapping by using weaving models to annotate the conceptual model before launching the transformation.

To conclude, we would like to point out two very important advantages of M2DAT-DB over existing works on DB schema design.

On the one hand, it supports a pure conceptual data model. In fact, existing tools start from a conceptual, in general an E/R model. However, they only support an adapted version of E/R ready to be mapped into logical models. For instance, they do not support n-ary relationships since they are challenging to be mapped to a relational model. By contrast, M2DAT-DB starts from a pure conceptual data model. It is the model transformations that deals with the problems inherent to the mapping of a conceptual data model to a logical one.

On the other hand, M2DAT-DB does support a standard data model. That is, existing tools focus on the logical model for specific products. In contrast, M2DAT-DB offers the possibility of generating the ORDB model conforming to the SQL:2003 standard. However, since no implementation conforms to the standard 100%, it allows moving from the standard logical model to a logical model for Oracle (probably the best DB engine, at least in what has to do with support for ORDB constructions). The use of a platform-specific model (logical model for SQL:2003) plus a platform-dependent model [322] (Oracle logical model) provides with much more flexibility and real interoperability between different DB vendors, since the SQL:2003 model acts as a pivot to/from which mappings from/to any particular logical model are much more simple.

All this given, from the point of view of MDE, where it is essential to rely on modelling tools as accurate and precise as possible, current technology for ORDB and XML schemas modelling is, at best, inadequate. M2DAT-DB aims at filling this gap.

1st Iteration: MIDAS-CASE

This thesis provides with a technical solution to build a framework for semi-automatic model-driven development of Web Information Systems. One of its main features is the use of existing technology in the field of MDE, like EMF, ATL, etc. Currently, this way of building frameworks for MDSD is probabbly the most adopted and it is gaining acceptance every day. Indeed, the State of the Art has shown a number of works following this approach, what serves to confirm that our decisions have proved to be right.

However, when we tackled the specification of a tool for MDSD of WIS, this type of decissions was not even considered since such MDE tools or components were just starting to emerge. The tendency was to build ad-hoc solutions using GPLs. Hence, in order to evaluate the advantages and drawbacks of this approach, we addressed the specification and construction of a stand-alone CASE tool for MDSD of WIS: MIDAS-CASE.

This chapter presents MIDAS-CASE specification and the prototypes that provides with a reference implementation. Likewise, the lessons learned from MIDAS-CASE project are put forward since they will be used in Chapter 4 to justify some of the technical decisions collected in M2DAT specification.

Finally, it is worth mentioning that, though MIDAS-CASE suffers from the technical limitations already sketched, its conceptual design is still valid and thus partially reflected on M2DAT conceptual architecture. Indeed, the mandatory requirement of modularization promotted by MIDAS was the driving force in the specification of MIDAS-CASE architecture. Such premise was preserved in M2DAT.

3.1 MIDAS-CASE: a stand-alone CASE tool for MDSD of WIS

There were some objectives to meet when we first faced the task of developing a supporting environment for the MIDAS methodology. Basically, we had to develop an environment that supports the graphical representation of all the models comprised in MIDAS, the automatic mapping between them and the automatic code generation from those models. Moreover, we planned to use an XML DB repository for XML-based storage of the models.

From the very first moment, we considered two requirements that technical support for MIDAS had to meet: it had to be both easily scalable and highly modular.

The aim was to be able to support a constant evolution of the tool. Since it was thought as a research prototype, new functionalities were to be added day by day. Furthermore, those already supported were candidates for continuous improvement, refinement. Besides, we aim at supporting the rapid inclusion of emerging technologies.

The result of MIDAS-CASE project wvere mainly:

- The definition of **MIDAS-CASE architecture**, which constitutes the basis of the actual version of M2DAT architecture.
- **MIDAS-CASE4WS** and **MIDAS-CASE4XS**. Two prototypes of MIDAS-CASE modules. MIDAS-CASE4WS uses the UML extension for WSDL (Web Services Description Language) [383] proposed in [225] to support the modelling of Web Services in extended UML and the automatic generation of the respective Web Service description in the WSDL standard. On the other hand, MIDAS-CASE4XS uses the UML extension introduced in [364] to support the modelling of XML Schemas in extended UML. From that model, the tool generates the corresponding XML Schema. The former prototype was presented in [353] whereas the later was introduced in [354].

MIDAS argued in favour of using UML.Specifically, a class diagram is used to depict conceptual data models. Thus, we also developed a module for UML class diagrams in MIDAS-CASE to ensure the integration with the rest of MIDAS-CASE models (UML extended models). To do so, we opted for having our own UML editor in MIDAS-CASE. We will not go deep into this editor since a pure UML editor is not a real contribution (there are quite a lot of commercial products for this task in the market). However, we include some screen captures on the appendix, next to the corresponding metamodel for class diagrams.

3.2 MIDAS-CASE Architecture

MIDAS-CASE architecture, shown on Figure 3-1, was defined according to two orthogonal dimensions.

On the one hand, it assumed a classical three-tier architecture [119], considering the traditional aspects in software development, i.e. the user interface, the logic and the persistence or data tiers.

On the other hand, MIDAS-CASE architecture was composed of a set of modules or subsystems, one for each concern considered in MIDAS for the development of the WIS (the hypertext, the content, the semantics, etc.). This way, support for new concerns could be added during the life cycle of the project, either as a response to new requirements or just to incorporate new advances in the field. The result is a scalable and easy-to extend tool. The support for a new model is embedded in a new module.

Figure 3-1 shows the original MIDAS-CASE Architecture including three different modules. From left to right, dotted rectangles distinguish the Object-Relational, the Web services and the *Others* module. The latter tries to show how the rest of subsystems are to be integrated in MIDAS-CASE to extend its initial capabilities.

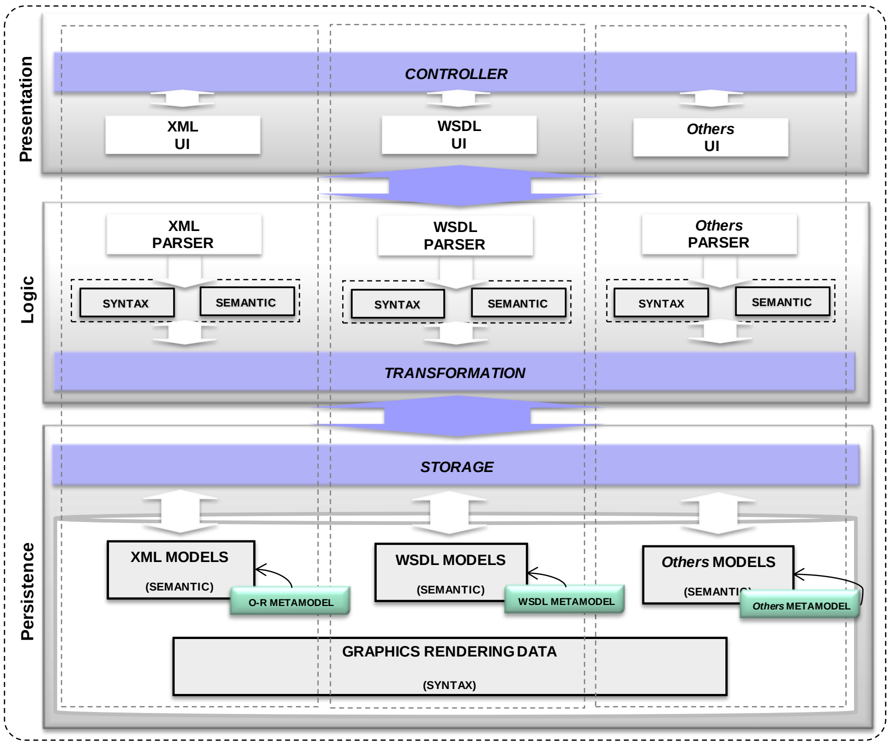


Figure 3-1. MIDAS-CASE Architecture

The whole architecture is guided by a common idea: MIDAS-CASE allows the definition of extended UML models that are stored in XML format. To carry out this task, the metamodel for each type of model considered in MIDAS is specified in an XML Schema. This way, the XML document that persists the model conforms to the respective XML Schema. If the XML document does not conform to the Schema, we conclude that the model is not valid.

In the following we summarize the main features of each tier from top to bottom.

3.2.1 Presentation

The presentation tier corresponds to the user interface. It encodes both the graphical representation of the model depicted on the working panel, and the controls to add, delete or modify any element from the model. In turn, it is composed of two layers.

First, a common unit or module, so-called *controller*, comprises the controls common to any of the different modules of MIDAS-CASE. That is, those controls that are shown despite the kind of model depicted in the working panel. This module has to load one of the lower *UI* (*User Interface*) units. They customize the user interface according to the model being edited. In other words, it provides with the graphic controls for the elements included in the corresponding metamodel. Thus, there is a different UI unit for each module.

3.2.2 Logic

Again, the logic tier comprises two different levels: in the upper level there is a set of parsers, one for each kind of model. From the information conatined in the diagram, they generate two XML documents to store seprately the semantics and the syntax of the model. On the other hand, the lower level is a common module to support transformations between the different types of models.

3.2.2.1 Parsers

All the parsers share the same internal architecture and perform the same task. Each one collects the data provided by the corresponding UI unit to generate two XML documents. The first one gatthers the semantics of the model depicted in the working panel. The other one collects the syntax or layout of the diagram. That is, the position and dimensions of the graphical elements included that compose the diagram. This distinction serves to separate the relevant from the secondary information.

For instance, in a conceptual data model we are interested in which the attributes of a class are, whether o not there is an association between two classes, etc. Nevertheless, we are not interested in the size of the rectangle that represents the class, nor the shape of the line representing the association. The XML document containing the semantics of the model is the one used for *model processing* (we use this term to refer to all the ways in which a model can be processed: validated, transformed, generated into code, interpreted, etc. [368])

We used a variation of GXL (Graph eXchange Language) to serialize the syntax of the model. GXL [167, 381] was a standard XML format for data interchange between graph-based tools. Starting from the GXL proposal, defined as a DTD, an XML Schema was defined ad-hoc for the special nature of MIDAS-CASE. Such Schema defines the structure of the XML document used to collect the syntax of any MIDAS-CASE model.

3.2.2.2 Transformation

We need to connect the different modules that support the development of each concern of the WIS. In other words, we need to connect the models depicted with each module. To that end, the transformation bus has to implement the mappings between those models. However, model transformation was still a bedding research field by the time MIDAS-CASE was designed. Hence, we adopted a very simplistic approach: we just encode the mapping rules for each specific transformation in the tool, i.e. we code them with the GPL used to build MIDAS-CASE (JAVA). Later on, we will discuss the problems derived from such approach.

3.2.3 Persistence

Since XML was gaining acceptance as format for data management and storage, the models created with MIDAS-CASE are serialized as XML documents. Therefore, after studying the different solutions for XML content management [81, 378], we opted for using an XML DB as the underlying repository of MIDAS-CASE, specifically, Oracle XML DB [117, 295].

Oracle XML DB was the first attempt of Oracle to support native XML storage and management. Its basis was a flexible mapping of XML Schemas to object-relational database schemas. Then, any XML document stored on the DB is shredded among the DB objects that mapped the corresponding XML Schema. We exploit this feature in MIDAS-CASE. Each XML Schema used to define the syntax and semantics of the models supported by MIDAS-CASE were mapped to an Oracle DB schema. This way, as shown in Figure 3-1, the basis of MIDAS-CASE models repository is an Oracle XML DB. Next section summarizes the management process for such repository.

3.2.3.1 Using an XML Database as Models Repository

The first step in the management process of the models repository of MIDAS-CASE is the specification of the metamodels of each type of model supported by the tool (step (a) in Figure 3-2). In essence, each module supports one or more models. Such metamodels are defined by means of an XML Schema.

Next, the set of XML Schemas are registered on the repository of Oracle XML DB (step (b) in Figure 3-2). As a result, a set of DB schemas is created [295, 347, 354] in the underlying DB. Each schema supports the storage of the models conforming to the respective metamodel. Notice that when we talk about models, we are referring just to the XML document that collects the semantics of the model. Nevertheless, recovering a MIDAS-CASE model also implies retrieving its syntax, that is, the rendering information of the diagram that depicts the model. As we have described in section 3.2.2.1, the syntax is stored in a separate XML document. The structure of those documents is the same for any kind of model. Thus, one XML Schema is enough to define it and thus one DB schema is enough to store the syntax of any model depicted with MIDAS-CASE, despite of the type of the model.

Finally, the XML DB that supports the MIDAS-CASE repository is composed of a set of DB schemas: one for every type of model supported, plus one for storing the syntax of any diagram (see Figure 3-1). Whenever a model is persisted, the logic tier generates the two XML documents (syntax and semantics) and passes it to the persitence tier. Then, the storage unit loads each document into the corresponding DB schema, i.e. the XML data is shredded into the corresponding OR tables. Likewise, whenever a model has to be retrieved, the two XML documents are passed from the persistence to the logic tier. This one uses the mutual references in the documents to build a diagram according to the syntax and semantics collected on those documents. Later, the UI unit depicts the diagram built.

The underlying XML Database provides with all the advantages inherent to DBs for the management of the models respository. For instance, retrieving all the models in which a particular class appears; modifying some information about a particular element on every model at a time; querying a model or just some specific parts of them, etc.

In addition, Oracle XML DB provides automatic validation of the models generated with MIDAS-CASE for free since each metamodel is defined by means of a (registered) XML Schema and every model is persisted as an XML document.

If the document conforms to the respective Schema, the model is said to be valid according to the respective metamodel.

Regarding the mapping rules, the idea is similar. Once a model is obtained by applying the mapping rules over a source model, the target model is validated as described above. If the model is valid, we infer that so they were the mapping rules.

XML DB Storage

Figure 3-2 summarizes the internals of using Oracle XML DB for models storage.

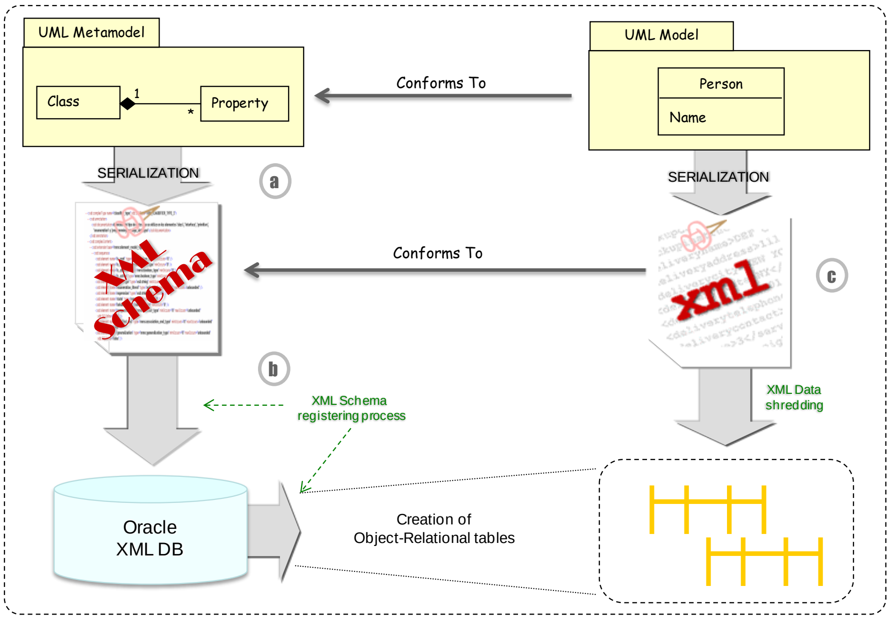


Figure 3-2. The use of an XML Database as models repository

First, the metamodel is serialized into an XML Schema (a). Such Schema is registered in XML DB (b). As a result, a new DB schema is created containing a set of user-defined Data types and typed tables. The registration is equivalente to launch a SQL script that creates the mentioned objects.

From that moment on, whenever the DB receives an XML document (c), it checks the XSD Uri of the document against those from the XML Schemas already registered. Then the XML data is shredded into the tables corresponding to the XML Schema matched.

3.3 MIDAS-CASE Prototypes

To prove that the architecture of MIDAS-CASE was a valid and feasible proposal, we developed a couple of prototypes: MIDAS-CASE4WS and MIDAS-CASE4XS. Both of them follow the same idea: they implement one of the UML profiles of MIDAS and support code generation from a UML stereotyped model.

The former uses the UML extension for WSDL proposed in [225] to support the modelling of Web Services in extended UML. From this model, the tool generates the description of the Web Service in the WSDL standard.

The later implements a refined version of the UML extension for XML Schema from [364]. From such UML extended model, the tool generates the corresponding XML Schema.

Both profiles have been defined following the steps described in [147] for the definition of UML profiles: first, identifying the elements of interest from the domain to model and collecting them into a metamodel; next, extending the UML standard to cope with the previous metamodel.

In the following, we present both prototypes by means of a pair of case studies. Previously, we introduce the UML extensions that they implement.

3.3.1 Modelling Web Services with Extended UML

As previously mentioned, in order to be able to model Web Services with UML, we have to define a Web Services metamodel This task had been solved previously by the WSDL standard that states which the components needed to precisely describe a Web Service were. Therefore, we went to the specification [383] to extract the main constructors identified by WSDL to describe a Web Service. The result is the Web Services metamodel presented in the following section.

Once we had defined the metamodel, we need to extend the UML standard to cope with the previous metamodel.

3.3.1.1 WSDL Metamodel

Figure 3-3 shows the WSDL meta-model represented by an UML class diagram. The definitions included on the WSDL standard can be separated into two different groups of components, according to the abstraction level of the respective concept represented by each class.

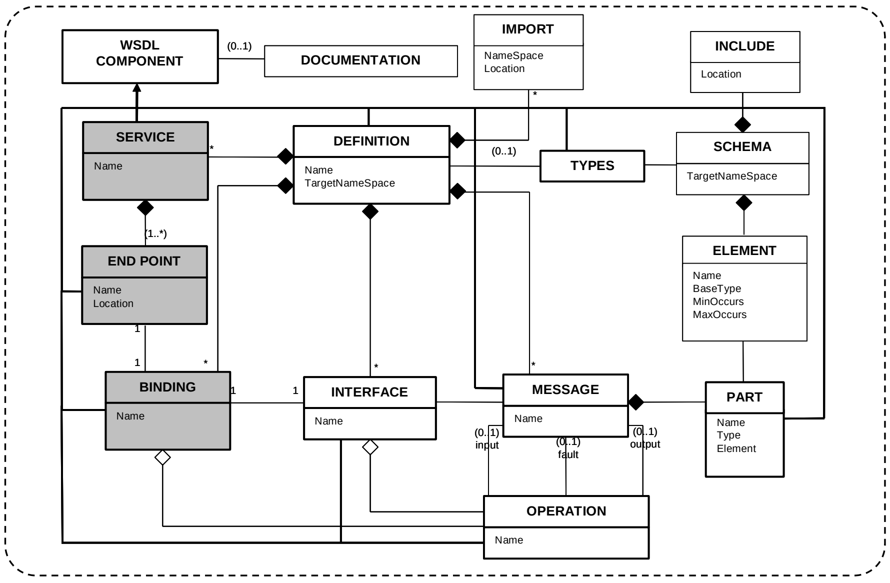


Figure 3-3. WSDL Metamodel

- The operations offered by a Web Service are grouped in *INTERFACEs*. Every time an *OPERATION* is used, the requester interchanges a set of *MESSAGEs* with the service provider. At the same time each message can contain several *PARTs* or parameters, whose type can be a base type XSD [385] (int, float, string, etc.), or any one of the types defined in the types section. In the last case, the data type can be defined by means of a *TYPE* or an *ELEMENT* attribute, from one of the *SCHEMAs* referenced in the WSDL document.
- A Web Service could be defined with the components already mentioned, despite of the platform or the language used for implementing it. The *BINDING* component allows binding the conceptual Web Service definition with the implementation in a specific platform. Each implementation will offer access points (*END POINT*) to the whole *SERVICE*.

Please, refer to the enclosed CD to find the XML Schema used to specify this metamodel.

3.3.1.2 UML Profile for WSDL

After defining the metamodel, we have to identify which UML metaclasses had to be extended and how to do it. To this end, the following design guidelines were defined in [225]:

- *DEFINITION* objects will be represented by means of a stereotyped class since it is explicitly defined in the WSDL specification and it constitutes the root of the document. All other elements will be nested within.
- Likewise, *MESSAGE*, *PART*, *INTERFACE*, *OPERATION*, *BINDING*, *ENDPOINT*, *SERVICE* and *IMPORT* components have been considered stereotyped classes since they are essential components, explicitly defined in the WSDL specification.
- TYPES and SCHEMA components will be represented by means of stereotyped compositions (<<*Type Schema*>>). Those elements represent the relation between a DEFINITION component and its data type definitions.
- Each *PART* component will be related to the *MESSAGE* component using it by means of a composition.
- The *PART-ELEMENT* relationship will be represented as a stereotyped association. If the *PART* component is used as a type, the stereotype will be <<*Part Type*>>, else the stereotype will be <<*Part Element*>>.
- The relationship between an *OPERATION* component and the *MESSAGEs* that it uses will be represented as an association stereotyped with <<*In*>>, <<*Out*>> or <<*Fault*>>, depending on the type of the message: an *input message*, an *output message* or a *fault message*.
- The *MESSAGE*, *INTERFACE*, *BINDING*, *SERVICE* and *IMPORT* components will be related with the *DEFINITION* component by means of a composition.

Finally, WSDL data types are based on the XML Schema Standard. Thus, we use the UML extension for XML Schemas modelling proposed in [364]. All things considered, the resulting UML profile is graphically depicted on Figure 3-4.

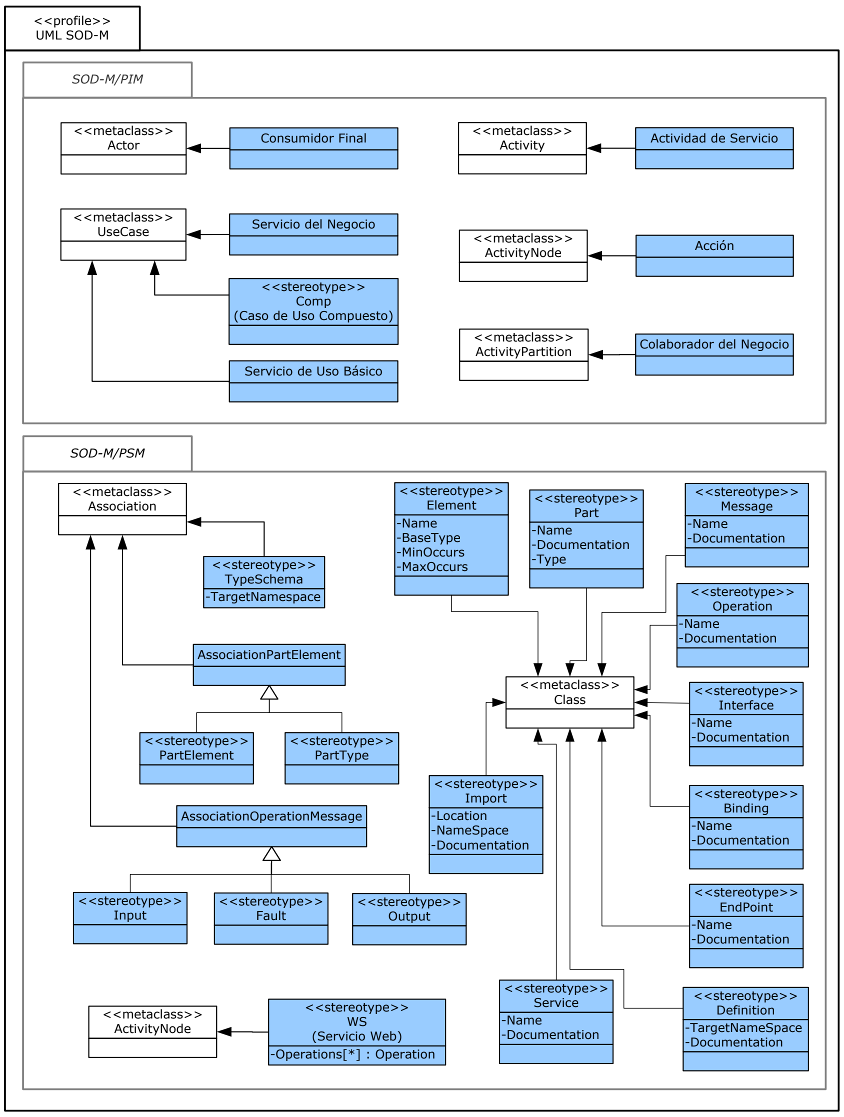

<<profile>>
UML SOD-M
SOD-M/PIM
<<metaclass>> Actor
Consumidor Final
<<metaclass>> Activity
Actividad de Servicio
<<metaclass>> UseCase
Servicio del Negocio
<<metaclass>> ActivityNode
Acción
<<stereotype>> Comp (Caso de Uso Compuesto)
<<metaclass>> ActivityPartition
Colaborador del Negocio
Servicio de Uso Básico
SOD-M/PSM
<<metaclass>> Association
<<stereotype>> Element -Name -BaseType -MinOccurs -MaxOccurs
<<stereotype>> Part -Name -Documentation -Type
<<stereotype>> Message -Name -Documentation
<<stereotype>> TypeSchema -TargetNamespace
<<stereotype>> Operation -Name -Documentation
AssociationPartElement
<<metaclass>> Class
<<stereotype>> Interface -Name -Documentation
<<stereotype>> PartElement
<<stereotype>> PartType
<<stereotype>> Import -Location -NameSpace -Documentation
<<stereotype>> Binding -Name -Documentation
AssociationOperationMessage
<<stereotype>> EndPoint -Name -Documentation
<<stereotype>> Input
<<stereotype>> Fault
<<stereotype>> Output
<<stereotype>> Service -Name -Documentation
<<stereotype>> Definition -TargetNameSpace -Documentation
<<metaclass>> ActivityNode
<<stereotype>> WS (Servicio Web) -Operations[*] : Operation

Figure 3-4. UML Profile for Web Services Modelling

The UML extension collects a set of stereotypes, tagged values and constraints that enable us to describe a Web Service according to WSDL standard and using UML as notation. In the following, we show how we use this extension in a case study.

3.3.2 MIDAS-CASE4WS Case Study: a Web Service for validating e-mail addresses

This section shows the application of the profile just introduced to model a Web Service for validating e-mail addresses. This is the case study that will be used to introduce MIDAS-CASE4WS.

Figure 3-5 shows the WSDL document for describing the *ValidateEmail* Web Service.

```
<?xml version="1.0" encoding="UTF-8"?>
<UMLWSDL2.0testing-Diagram xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
 xsi:noNameSpaceSchemaLocation="http://kybele.escet.urjc.es/UMLWSDL2.02Schema-testing">
 <definitions>
  <targetNameSpace>http://example.com/ValidateEmail</targetNameSpace>
  <nameSpace location="http://schemas.soap.org/wsdl" prefix="default"/>
  <nameSpace location="http://www.w3.org/2001/XMLSchema" prefix="xs"/>
  <nameSpace location="http://schemas.xmlsoap.org/wsdl/soap" prefix="soap"/>
  <type name="ValidateResponse" complexity="complexType"/>
  <type name="ValidateAddress" complexity="complexType"/>
  <typeSchema>
   <targetNameSpace>http://soap.einsteinware.com/Email</targetNameSpace>
        <type>ValidateAddress</type>
        <type>ValidateResponse</type>
  </typeSchema>
  <message name="ValidateEmailAddressSoapOut">
   <part name="ParametersIn" typeOfPart="udtType">
    <udtType>ValidateAddress</udtType>
   </part>
  </message>
  <message name="ValidateEmailAddressSoapIn">
   <part name="ParametersOut" typeOfPart="udtType">
    <udtType>ValidateResponse</udtType>
   </part>
  </message>
  <interface name="EmailServiceInterface">
   <operation name="ValidateEmailAddress">
    <input name="ValidateEmailAddressSoapOut"/>
    <output name="ValidateEmailAddressSoapIn"/>
   </operation>
  </interface>
  <binding name="EmailServiceBinding" type="EmailServiceInterface">
   <soap-binding protocol="http://schemas.xmlsoap.org/soap/http"/>
   <operation-binding name="ValidateEmailAddress">
    <soap-operation style="encoded“
              soapAction="http://soap.einsteinware.com/emailservices.aspx/ValidateEmailAddress"/>
    <input>
     <soap-body use="encoded" namespace=http://soap.einsteinware.com/emailservices.aspx
              encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
    </input>
    <output>
     <soap-body use="literal"/>
    </output>
   </operation-binding>
  </binding>
  <service name="ValidateEmailService">
   <endpoint name="EmailServiceSoap" binding="EmailServiceBinding">
    <soap-address location="http://soap.einsteinware.com/emailservices.aspx"/>
   </endpoint>
  </service>
</definitions>
</UMLWSDL2.0testing-Diagram>
```

Figure 3-5. WSDL description of the ValidateEmail Web Service

The Web Service defines one operation, *ValidateEmailAddress*, owning two messages (input and output). The input message, *ValidateEmailAddressSoapIn* defines one part, *ParametersIn*, whose type is defined by the *ValidateEmailAddress* XML Element. Likewise, the output message *ValidateEmailAddressSoapOut* defines the *ParametersOut* part whose data type is the *ValidateEmailResponse* XML Element. Both XML Elements are located at the *Types* section since they are included in the WSDL document just for data typing purposes.

The portType *EmailServicePortType* collects the operations that will be performed by the Service. In this case there is only one operation, *ValidateEmailAddress*. The link between this portType and the SOAP protocol is described by the *EmailServiceBinding* Element. The service has only one port *EmailServiceSoap*, which defines the Web Service location through an URL.

Figure 3-6 shows the *ValidateEmail* Web Service modeled using the mentioned UML extension with MIDAS-CASE4WS.

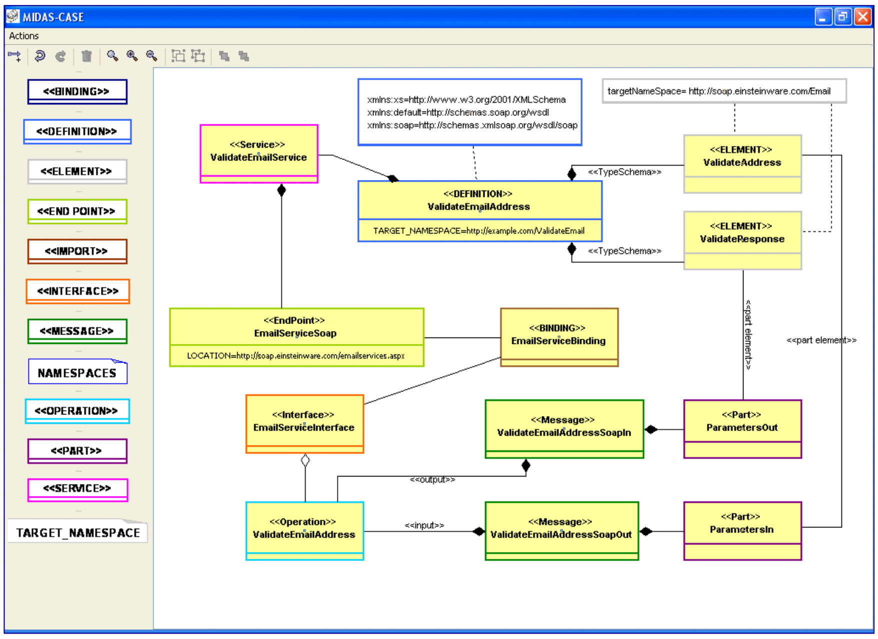

MIDAS-CASE
Actions
<<BINDING>>
<<DEFINITION>>
<<ELEMENT>>
<<END POINT>>
<<IMPORT>>
<<INTERFACE>>
<<MESSAGE>>
NAMESPACES
<<OPERATION>>
<<PART>>
<<SERVICE>>
TARGET_NAMESPACE
xmlns:xs=http://www.w3.org/2001/XMLSchema
xmlns:default=http://schemas.soap.org/wsdl
xmlns:soap=http://schemas.xmlsoap.org/wsdl/soap
targetNameSpace= http://soap.einsteinware.com/Email
<<Service>> ValidateEmailService
<<DEFINITION>> ValidateEmailAddress
TARGET_NAMESPACE=http://example.com/ValidateEmail
<<TypeSchema>>
<<ELEMENT>> ValidateAddress
<<ELEMENT>> ValidateResponse
<<TypeSchema>>
<<EndPoint>> EmailServiceSoap
LOCATION=http://soap.einsteinware.com/emailservices.aspx
<<BINDING>> EmailServiceBinding
<<part element>>
<<part element>>
<<Interface>> EmailServiceInterface
<<Message>> ValidateEmailAddressSoapIn
<<Part>> ParametersOut
<<output>>
<<Operation>> ValidateEmailAddress
<<input>>
<<Message>> ValidateEmailAddressSoapOut
<<Part>> ParametersIn

Figure 3-6. Screen Capture from MIDAS-CASE4WS: *ValidateEmail* Web Service represented in extended UML

The working panel depicts the Web Service model for the above case study. The controls bar on the left hand allows adding any one of the elements identified in the presented WSDL metamodel. The upper tools bar provides with the usual utilities, such as undo/redo, zoom in/zoom out, etc.

Back to the model itself, the *name* attribute of the definition component serves to name the class while the *TargetNameSpace* attribute will be represented as a property of the class.

The set of namespaces used to build the WSDL document are represented as tagged values. For the sake of clarity, tagged values are represented as notes linked to the corresponding element. The relationship between the <<*DEFINITION*>> class and the data types *ValidateEmailAddress* and *ValidateEmailResponse* are represented by means of a composition stereotyped as <<*TypesSchema*>>. The *TargetNameSpace* attribute of the schema is be represented as a tagged value.

Regarding messages, they are connected to their parts by means of compositions, while <<*Part_Element*>> associations serve to connect each part with the data type it uses. Therefore, the association between *ParametersIn* part and *ValidateEmailAddress* element is stereotyped with <<*Part_Element*>>. And so it is the association between *ParametersOut* part and *ValidateEmailResponse*.

Next, the *EmailServicePortType* uses one operation, *ValidateEmailAddress*. This fact is represented by means of an aggregation between them. In turn, the operation defines two messages, *ValidateEmailAddressSoapIn* and *ValidateEmailAddressSoapOut*. Stereotyped associations represent this fact. The stereotype depends on the nature of the message (whether it is an <<*Input*>> or an <<*Output*>> message).

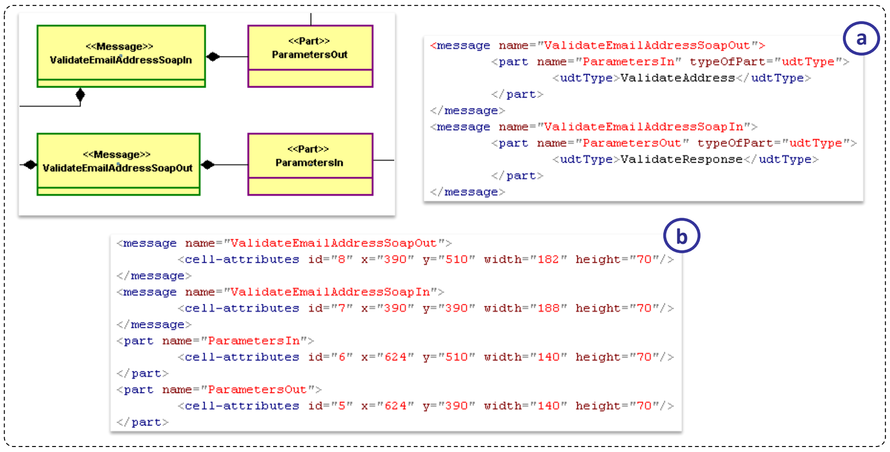
Whereas the binding is also included in the model, its connection with the SOAP protocol is omitted. The *EmailServiceBinding* component describes the binding to the porttype. Therefore an association is depicted connecting both of them.

Finally, the service and port elements are represented by means of the *EmailService* object that contains the *EmailServiceSoap* port. A composition is used to show this containment relation.

3.3.2.1 Distinguishing syntax from semantics

In previous sections, when we have described MIDAS-CASE architecture, we have mentioned that two different files were used to store any MIDAS-CASE model: one for the semantics and one for the syntax. This idea remains valid in the context of MDE (with some modifications as we will show). Hence, we elaborate more on this topic in this section. To that end, we use the case study to show the decomposition conducted between the syntax and the semantics of the model.

We focus on the definition of the *operation* offered by the Web service, the *messages* it uses, and the parameters or *parts* of each message. Left hand side of Figure 3-7 shows such messages and parts, while right hand side shows part of the XML files to store separately the semantics (a) and the syntax (b) of the model.



```
<message name="ValidateEmailAddressSoapOut">
        <part name="ParametersIn" typeOfPart="udtType">
                <udtType>ValidateAddress</udtType>
        </part>
</message>
<message name="ValidateEmailAddressSoapIn">
        <part name="ParametersOut" typeOfPart="udtType">
                <udtType>ValidateResponse</udtType>
        </part>
</message>
```

```
<message name="ValidateEmailAddressSoapOut">
        <cell-attributes id="8" x="390" y="510" width="182" height="70"/>
</message>
<message name="ValidateEmailAddressSoapIn">
        <cell-attributes id="7" x="390" y="390" width="188" height="70"/>
</message>
<part name="ParametersIn">
        <cell-attributes id="6" x="624" y="510" width="140" height="70"/>
</part>
<part name="ParametersOut">
        <cell-attributes id="5" x="624" y="390" width="140" height="70"/>
</part>
```

Figure 3-7. Excerpt from the *Validate E-Mail* Web Service model and XML files

On the one hand, the semantics of the Web Service is collected as a WSDL document. So, the data in the XML fragment showed in (a) is merely WSDL information. It describes the messages used by the operation provided by the Web Service and the parameters used by these messages.

On the other hand, the XML fragment in (b) contains just the data for the appropriate rendering of the drawing elements in the diagram. Therefore, an XML element stores the data for the element representing the input message; another one does so for the output message, etc.

3.3.3 *Modelling XML Schemas with Extended UML*

As it happens with Web Services, UML has to be extended to represent XML Schemas. Therefore, to develop MIDAS-CASE4XS, that aims at supporting XML Schema modelling with UML, we started from the UML profile proposed in [364, 365]. Following section summarizes the metamodel as well as the corresponding UML profile.

3.3.3.1 XML Schema Metamodel

Current M2DAT-DB bundles a complete DSL for XML Schemas modelling (including the corresponding metamodel). However, when we first addressed the task of supporting the modelling of XML Schemas, we sketched the one below. Although it is quite simple and obviously far from being complete, it was a good starting point and the origin of the final version that has been obtained after several iterations. As a matter of fact, a metamodel is not proven to be valid until you really start working with it. This implies not only creating terminal models conforming to your metamodel, but also coding model transformations and code generation programs that use it, either as source or target metamodel. Hence, the strengths and weaknesses of the initial version started to arose when we adressed the development of the tooling for the proposal, never before.

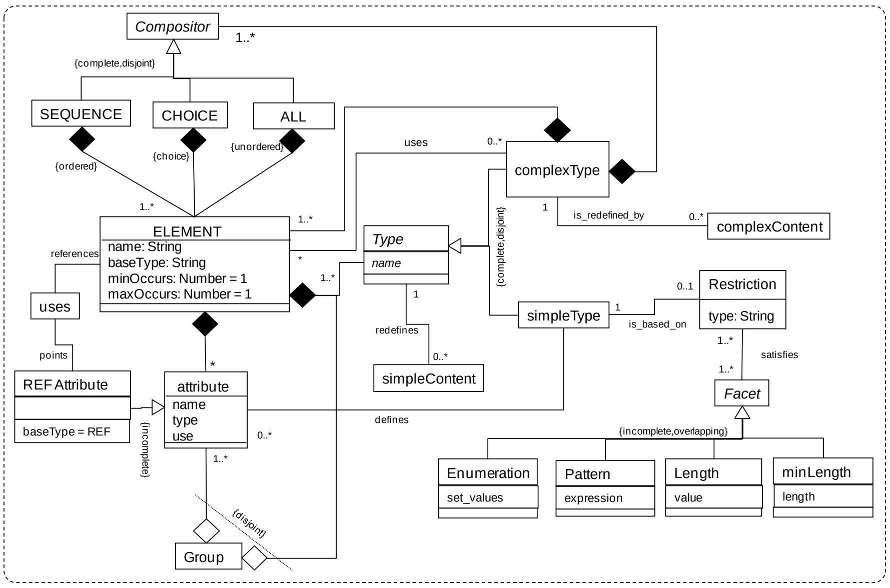


Figure 3-8. XML Schema metamodel and corresponding UML profile

In the following section we describe the metamodel next to the UML extension.

3.3.3.2 UML Profile for XML Schema

Again, for the definition of the UML extension for XML Schema modelling, we followed the guidelines proposed in [147].

First we defined the metamodel (previous subsection), using UML. Then, for each relevant element of the metamodel we included a stereotype in the profile. Notice that not all of the elements in the metamodel are “relevant elements”. Some of them can be ignored, or just represented by common UML elements. The UML extension is depicted in below.

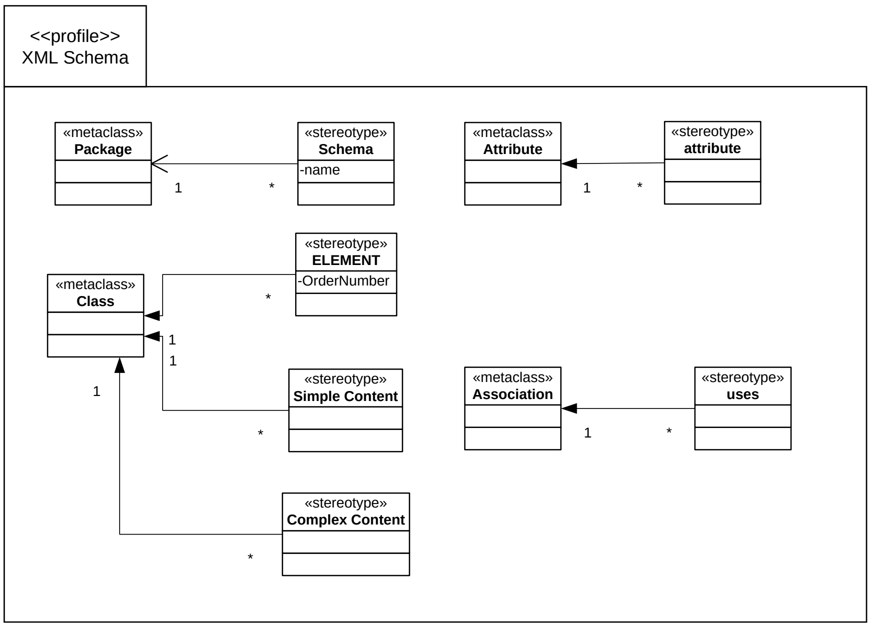


Figure 3-9. UML extension for XML Schema

- According to the proposed UML extension, an XML schema is represented by means of a UML package stereotyped with <<Schema>>, which will include all the components of the XML schema. The name of the schema will be the name of the package.
- The XML ELEMENTS are represented as classes stereotyped with <<ELEMENT>>. To name them, the 'name' attribute of the element is used, since they are explicitly defined in the XML Schema. The attributes of the element will be tagged values of the class. Besides, the order of appearance of the element in the including XML Schema is represented prefixing the name of the class.
- The XML ATTRIBUTES are represented by means of UML attributes added in the class that represents the containing XML element. The base type of an XML attribute will be represented as the data type of the corresponding UML attribute. The constraints to be satisfied by the attribute (required, optional) and the default or fixed value will be represented as tagged values.
- A COMPOSITOR composition is a special kind of composition stereotyped with the kind of compositor: <<Choice>>, <<Sequence>> or <<All>>. It can only be used to join an element (*whole*) with the elements that compose it

(*parts*). The compositors can be used to represent graphically nameless XML complexTypes.

- Named COMPLEX TYPES were represented as classes with the <<complexType>> stereotype. The complexType will be related by means of a USES association with the element, complexType or simpleType that uses it. If the complexType has no name, it will be represented in an implicit way by the compositor used to define the complexType.
- An XML type with no sub-elements or attributes is a SIMPLE TYPE. simpleTypes were considered as classes stereotyped with <<simpleType>>. Its name being the same of the containing element, to whom it will be related by means of a composition stereotyped with <<simpleType>>.
- An XML COMPLEXCONTENT element allows redefining a complexType. Thus, it was represented as a subclass of the complexType that it defines.
- A simple content element allows redefining an XML type. Therefore, SIMPLECONTENT elements were represented as stereotyped classes related by means of an inheritance association to the type (simple or complex type) which is redefined by the simpleContent type.
- A USES association is a special kind of unidirectional association, stereotyped as <<uses>>. It joins a named complexType with the element or type (simple or complex) that uses it. It can also be used to join two elements by means of a *ref* attribute in one of the elements. An arrow pointing to the referenced element represents the direction of the association.
- A REF element will be represented by means of an attribute stereotyped with <<REF>> and represents a link to another element.

3.3.4 MIDAS-CASE4XS Case Study: a Web Information System for medical images management

To show MIDAS-CASE4XS capabilities we briefly present a case study that is part of a real application presented in [364]: a Web Information System for medical images management. The system uses XML DB as data repository to integrate the management of both the structured (OR) and semi-structured (XML) data of the application.

Figure 3-10 shows the model for the XML Schema represented with extended UML that drives the design of the XML part of the DB

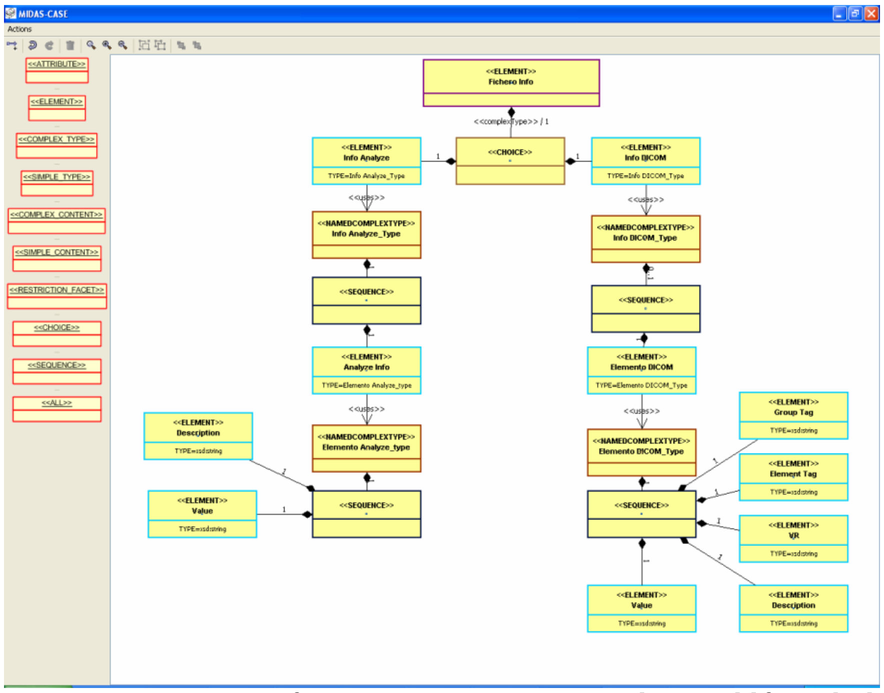

MIDAS-CASE
Actions
<<ATTRIBUTE>>
<<ELEMENT>>
<<COMPLEX_TYPE>>
<<SIMPLE_TYPE>>
<<COMPLEX_CONTENT>>
<<SIMPLE_CONTENT>>
<<RESTRICTION_FACET>>
<<CHOICE>>
<<SEQUENCE>>
<<ALL>>
<<ELEMENT>> Fichero Info
<<complexType>> / 1
<<CHOICE>>
<<ELEMENT>> Info Analyze TYPE=Info Analyze_Type
<<ELEMENT>> Info DICOM TYPE=Info DICOM_Type
<<uses>>
<<NAMEDCOMPLEXTYPE>> Info Analyze_Type
<<NAMEDCOMPLEXTYPE>> Info DICOM_Type
<<SEQUENCE>>
<<ELEMENT>> Analyze Info TYPE=Elemento Analyze_type
<<ELEMENT>> Elemento DICOM TYPE=Elemento DICOM_Type
<<NAMEDCOMPLEXTYPE>> Elemento Analyze_type
<<NAMEDCOMPLEXTYPE>> Elemento DICOM_Type
<<ELEMENT>> Description TYPE=xsd:string
<<ELEMENT>> Value TYPE=xsd:string
<<ELEMENT>> Group Tag TYPE=xsd:string
<<ELEMENT>> Element Tag TYPE=xsd:string
<<ELEMENT>> VR TYPE=xsd:string

Figure 3-10. Screen Capture from MIDAS-CASE4WS – XML Schema model for medical images management

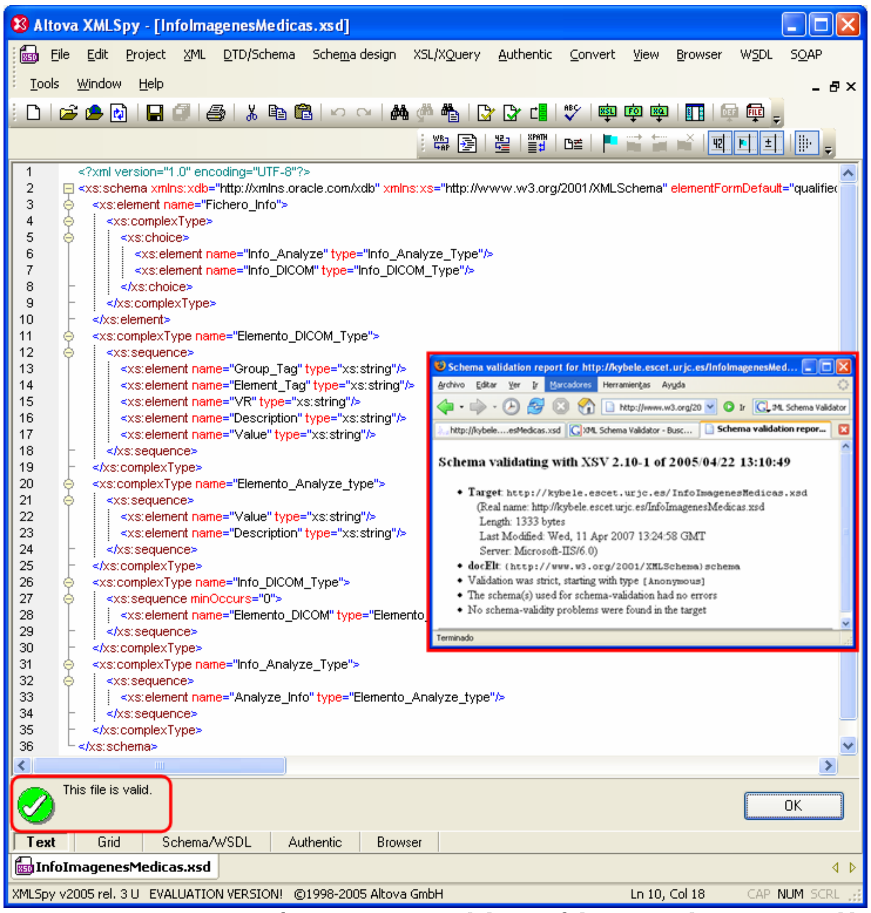
Broadly speaking, the information from each medical image is shredded into a group of *Fichero_Info* files. Each *Fichero_Info* could contain two types of data: Analyze (*Info_Analyze*) or DICOM (*Info_DICOM*), as stated by the *choice* compositor linked to the *Fichero_Info* element. Both, Analyze and DICOM are formats for medical images storing and interchanging [7, 230].

Intrun, each Analyze or DICOM element is composed of a set of elements whose internal structure differs from one to the other. To define such structures two named complex types are used, *Elemento_Analyze_Type* and *Elemento_DICOM_Type*. Both types are a *sequence* of XML elements.

3.3.4.1 Code Generation

As we have already mentioned, MIDAS-CASE modules provides with code generation from extended UML models. In the case of MIDAS-CASE4XS, the code generated is the XML document defining the XML Schema.

Back to the case study, the model shown in Figure 3-10 is automatically serialized into the XML Schema shown in Figure 3-11 shows such schema edited on XMLSpy [14], an XML editor that supports automatic validation of XML Schemas. Likewise, the output from validating the Schema at the official checker from the W3C (http://www.w3.org/2001/03/webdata/xsv) is also shown in the picture.


Altova XMLSpy - [InfoImagenesMedicas.xsd]
File Edit Project XML DTD/Schema Schema design XSL/XQuery Authentic Convert View Browser WSDL SOAP Tools Window Help
Schema validation report for http://kybele.escet.urjc.es/InfoImagenesMed...
Archivo Editar Ver Ir Marcadores Herramientas Ayuda
http://www.w3.org/20
XML Schema Validator
http://kybele.....esMedicas.xsd
XML Schema Validator - Busc...
Schema validation repor...
Terminado
This file is valid.
OK
Text Grid Schema/WSDL Authentic Browser
InfoImagenesMedicas.xsd
XMLSpy v2005 rel. 3 U EVALUATION VERSION! ©1998-2005 Altova GmbH
Ln 10, Col 18 CAP NUM SCRL

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xdb="http://xmlns.oracle.com/xdb" xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualifiec
  <xs:element name="Fichero_Info">
    <xs:complexType>
      <xs:choice>
        <xs:element name="Info_Analyze" type="Info_Analyze_Type"/>
        <xs:element name="Info_DICOM" type="Info_DICOM_Type"/>
      </xs:choice>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="Elemento_DICOM_Type">
    <xs:sequence>
      <xs:element name="Group_Tag" type="xs:string"/>
      <xs:element name="Element_Tag" type="xs:string"/>
      <xs:element name="VR" type="xs:string"/>
      <xs:element name="Description" type="xs:string"/>
      <xs:element name="Value" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="Elemento_Analyze_type">
    <xs:sequence>
      <xs:element name="Value" type="xs:string"/>
      <xs:element name="Description" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="Info_DICOM_Type">
    <xs:sequence minOccurs="0">
      <xs:element name="Elemento_DICOM" type="Elemento_
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="Info_Analyze_Type">
    <xs:sequence>
      <xs:element name="Analyze_Info" type="Elemento_Analyze_type"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Schema validating with XSV 2.10-1 of 2005/04/22 13:10:49

- **Target**: `http://kybele.escet.urjc.es/InfoImagenesMedicas.xsd`
  (Real name: http://kybele.escet.urjc.es/InfoImagenesMedicas.xsd
  Length: 1333 bytes
  Last Modified: Wed, 11 Apr 2007 13:24:58 GMT
  Server: Microsoft-IIS/6.0)
- **docElt**: `(http://www.w3.org/2001/XMLSchema)schema`
- Validation was strict, starting with type `[Anonymous]`
- The schema(s) used for schema-validation had no errors
- No schema-validity problems were found in the target

Figure 3-11. Screen Capture from XMLSpy - Validation of the XML Schema generated by MIDAS-CASE4XS

3.3.5 Developing MIDAS-CASE: Technical Issues

This section aims at summarizing some remarks about the implementation of MIDAS-CASE prototypes. Obviously, the three prototypes mentioned (MIDAS-CASE4WS, MIDAS-CASE4XS, MIDAS-CASE4UML) were developed using the same technologies, all of them spinning around the JAVA language. Next, we describe briefly some details about the implementation of each tier of MIDAS-CASE architecture.

3.3.5.1 Presentation

To develop the interface layer we used JGraph [10, 11]. JGraph is an open source graph component available for JAVA whose powerful API simplifies the tedious task of drawing diagrams. Figure 3-12 shows the JAVA classes architecture of MIDAS-CASE UI.

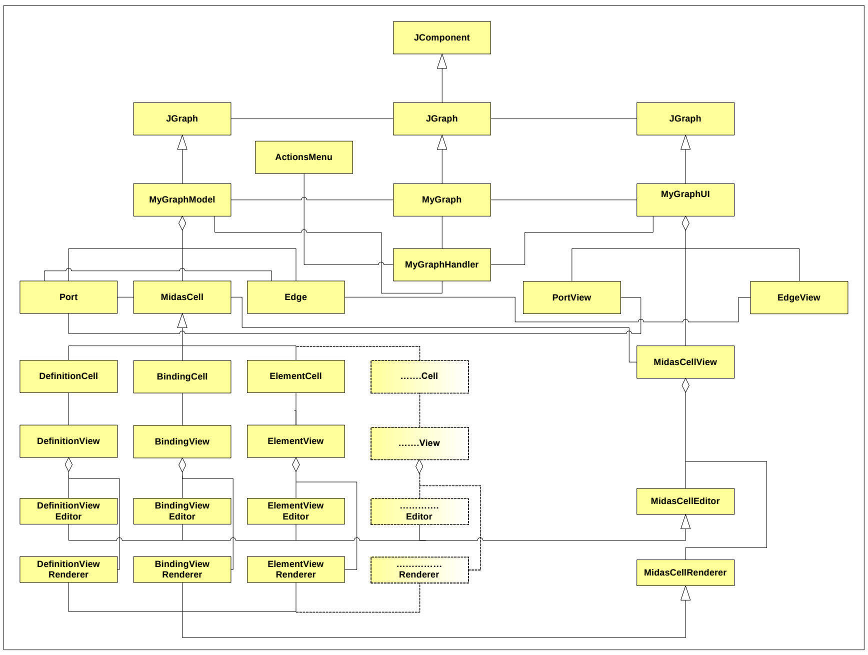


Figure 3-12. User Interface Layer Architecture

As it happens with almost every graphical application, it is based on the Model-View-Controller pattern (MVC, [148]). Each new model or diagram in MIDAS-CASE is a *MyGraph* object. This class extends *JGraph*, the root class of JGraph. The *MyGraph* object is associated with a data model (*MyGraphModel*) as well as a view (*MyGraphUI* object).

The data model is composed of a set of nodes and edges, plus ports. Each node owning at least one port. This way, two nodes are connected by connecting their ports. The data from each node lies in a *MidasCell* object while its presentation lies in a *MidasCellView* object. In turn, the aggregation of all the _*View* classes constitues the view of the model that is the *MyGraphUI* class.

For rendering the data, each *MidasCellView* is connected to a *Renderer* and an *Editor* object. The former deals with node appearance, such as dimensions, size, location, colours, shapes, etc. The later allows modifying the data of that node. That is, the information included in the corresponding *MidasCell* objetc.

Finally, event handling lies on the *MyGraphHandler* class, that connects the data model with its view.

3.3.5.2 Logic and Persistence

As explained in sections 3.2.2 and 3.2.3, each model created in MIDAS-CASE is persisted in two different XML documents (syntax and semantics). In some sense, we are separating the data model from its view. So, the conceptual architecture of MIDAS-CASE fits perfectly with the technical deployment of the tool based on JGraph.

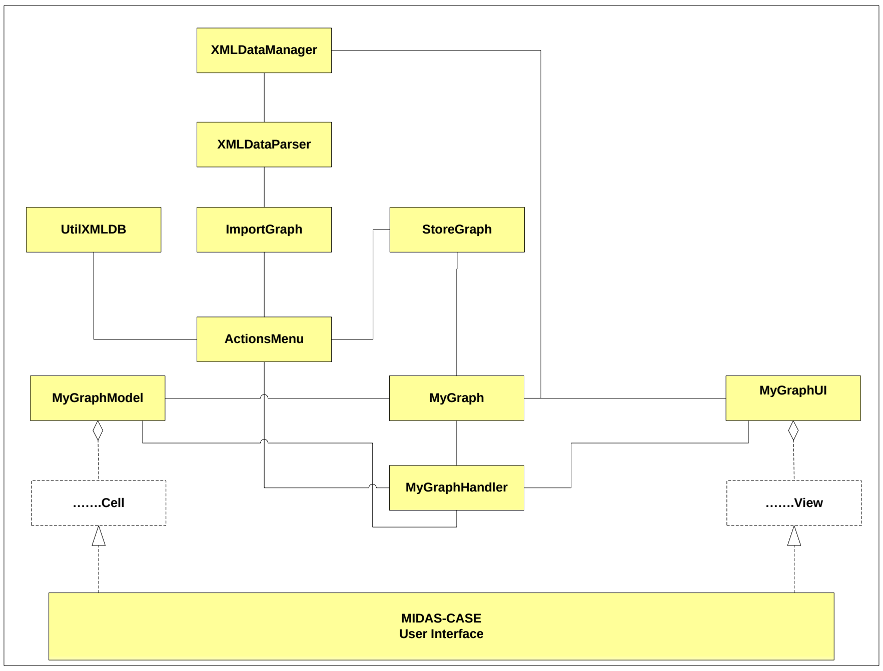


Figure 3-13. Application Logic and Persistente layers architecture

Figure 3-13 summarizes the JAVA classes architecture for the logic and persistence tiers. The *ActionMenu* class is connected to the *StoreGraph*, *ImportGraph* and *UtilXMLDB* classes. Whenever the user wants to save a model, the *ActionMenu* class handles the event raised. This one invokes the method of one of the other two classes.

When the user saves a model, the *StoreGraph* uses its link with the *MyGraph* class to collect all the information about the model. Next, it generates two XML documents: one conforming to the XML Schema for diagram data management and the other one conforming to the corresponding XML Schema, depending on the type of the model. That is, the XML Schema for persisting WSDL models XML Schema models or UML class diagrams. All these Schemas can be found on the enclosed CD.

Later on, an *UtilXMLDB* object is in charge of saving the XML documents in the Oracle XML DB. To do so, this class provides with methods for storing, retrieving, deleting and modifying XML documents stored in the DB.

On the other hand, it is the *ImportGraph* class, which deals with retrieving models. To do so, it creates a new *XMLDataParser* object that implements a SAX (Simple API for XML) parser. The XMLDataManager class handles the events raised by the parser as it finds XML tags. Handling these events means adding new elements to the model, as well as setting their properties. This information is found on the XML document being parsed.

3.3.6 Adding more Functionality to MIDAS-CASE

All along this chapter we have emphasized the relevance of extensibility in any tool for MDSD. In particular, the modular and open nature of MIDAS to new advances in the field implied the need to design MIDAS-CASE as an open framework, ready to incorporate new technologies and support new fucntionalities.

Therefore, when we first planned the development of MIDAS-CASE we stated that the tool had to be both modular and scalable. The underlying idea was to make lighter the workload related with adding more functionality. The modular architecture of MIDAS-CASE helps on this task. So, whenever a new type of model has to be supported, a new module is developed. To buid the new module, we just have to follow the same architecture and development process of the previous ones. Hence, we need to identify clearly the different steps to carry out in order to complete the development process of the module.

In the following, we describe the set of steps to carry out such process. Note that they are clearly identified, what eases the task of developing a new module. Figure 3-14 summarizes these steps.

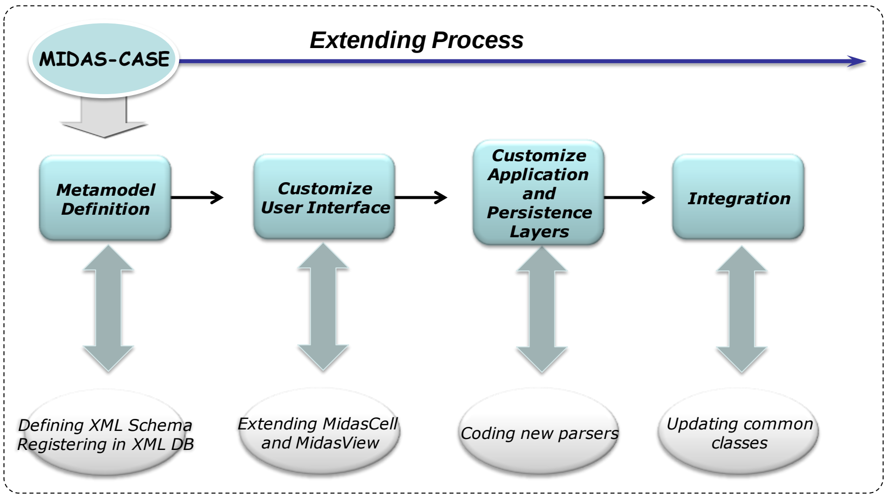


Figure 3-14. MIDAS-CASE Extending process

- First, the metamodel for the new type of model is defined in an XML Schema. Then, the Schema is registered in the underlying XML DB. This way, the repository is ready to store models conforming to the new metamodel.
- Next, we have to customize the user interface to add support for modelling the elements included in the new metamodel. To do so, we have to extend the *MidasCell* and *MidasView* classes (see section 3.3.5.2).
- To adapt both the logic and the persistence tiers to the new type of model, new *StoreGraph* and *XMLDataManager* classes are implemented. They provide with the parsers to retrieve and store models conforming to the recently added metamodel. Anyway, the gap between the classes to implement and the existing ones is minimal. The changes consist of a few modifications over some methods well-localized.
- Finally, minor revisions over the common classes (those implementing menus and toolbars) will integrate the newly implemented functionality into MIDAS-CASE.

3.4 Lessons Learned

MIDAS-CASE was a coarse approach to a tool for MDSD of WIS. Nevertheless, designing, planning and developing MIDAS-CASE provided with a set of lessons learned and good practices. Likewise, some of the ideas that we tried to capture in MIDAS-CASE had proven to be valid later. In fact, the most important MDE frameworks or tools are total or partially based on some of those ideas.

Next, we summarize and put forward some of them, next to the main conclusions obtained from the MIDAS-CASE project. We follow the same structure used to present MIDAS-CASE: we present these ideas in a top-down way, from the presentation to the persistence tiers.

3.4.1 User Interface Development

One of the main tasks that a tool supporting MDSD has to provide with is model editors. In fact, such editors constitute the most of the user interface of this type of tools. Therefore, mastering the development of model editors is a must in order to develop tools for MDSD.

In the context of the MIDAS-CASE project we have constated that the use of graphical components (like JGraph) to develop user interfaces increases the degree of freedom. You are able to do almost anything when coding the user interface from scratch since it provides with fine-grained control over the result. Taking it to the extremes, we might say that the graphical component provides with an API to create and handle (very simple) lines and nodes and it is up to the developer how they are combined in order to create the look and feel of the tool.

On the other hand we consider the frameworks for development of tools for MDSD. Some of them include facilities for developing graphical editors for models, starting from the corresponding metamodel (see EMF/GMF or MetaEdit). However, when using such frameworks you are losing control over the result. Developing the editor (at least, the graphical part) is easier, but the kind of things your editor will be able to do are limited by the capabilities of the framework. For instance, you may not be able to use a specific shape or it may be impossible to format the information shown on those elements as needed

Regarding MIDAS-CASE, the screen captures spread over the previous sections prove that such approach works well. The result is quite appealing and the editors built are intuitive and elegant.

Nevertheless, you should consider that coding the user interface was the most time-consuming task of MIDAS-CASE development. Given that there should be a compromise between the graphical capabilities you want for your tool and the effort you dedicate to implement the user interface, the lesson to learn is to use one of the above-mentioned facilities whenever you need to develop a models editor.

3.4.2 XML Schema as (meta)modelling language

A quick look at the most accepted modelling tools serves to confirm that the use of XML Schemas to express metamodels and XML documents to express conforming models, have been finally the most adopted way of building models repositories.

Indeed, UML models are persisted using XMI [391], the OMG standard for interchanging, manipulating and integrating XML data and objects. Hence, a tool implementing XMI creates an XML Schema from the UML model (before XMI 2.0, a DTD was created instead of the Schema) [386]. From there on, you may define terminal models conforming to your model by defining XML documents conforming to the above Schema.

EMF [382] itself follows this approach. Its underlying format for model management is also based on XMI. In a simplistic approach, we could argue that Ecore metamodels are direct translations of XML Schemas to (XMI) conforming Schemas. Likewise, Ecore terminal models are (almost) XML documents conforming to the above Schemas.

Since XML syntax is too verbose and not very easy to use, abstractions over the underlying models are provided in the form of graphical editors to simplify the task of model editing. However, the idea of using XML as modelling language remain valid, though it is done at lower abstraction levels.

3.4.3 Separating the Abstract Syntax from the Concrete Syntax

All along the previous sections we have stressed the separation between syntax and semantics that we implemented in MIDAS-CASE. In fact, we used different files to store the syntax and the semantics of each model.

At the time of writing this dissertation, when we revisited the MIDAS-CASE project we realised that the idea of distinguishing syntax from semantics is widely adopted by the most relevant MDE tools, though the terminology has changed (see section 2.1.4 for clear definitions of each term), maybe because we were on the dawning of MDE and some concepts were still a bit unclear. At present, **abstract syntax** is used to refer to what we called semantics in MIDAS-CASE. Likewise, we used just syntax instead of the actual **concrete syntax**.

However, the idea remains valid: we need to separate the relevant information from the way we render it. When talking about (graphical) models we mean that we distinguish the abstract syntax (the concepts modelled) from the concrete syntax (the layout of the visual diagrams used to show them).

In fact, this is one of the bases of EMF, where models are stored as *.Ecore* files whereas diagrams are stored as *.diagram* files. As well, OMG followed this approach when they liberated the UML Diagram Interchange (XMI-DI) specification [271]. They aimed at easing the task of model exchange between different modelling tools. Since XMI was devised just for exchanging the abstract syntax of models, the concrete syntax was lost. Thus, when a model was imported from other tool, it was not displayed correctly. Indeed, apart from the eternal dilemma around XMI versioning drawbacks, nothing was said about diagramming information exchange. Definitively, this fact hampered the adoption of XMI and thus, OMG conceived the UML Diagram Interchange to cover this gap.

Nevertheless, CASE tools developers did not pay a slight bit of attention on XMI-DI due to its limitations. Basically, it was not rich enough to support exchange of the visual presentation of a model. In fact, OMG has created recently a working group to facilitate interoperability among software models and model standards, i.e. to taddress specifically this issue, the Model Interchange Working Group (MIWG) [276].

The problem is still a serious drawback on the most accepted industrial tools. Just to cite an example, we have tried ERwin to ArgoUML import/export (see section 2.4.3.7). Indeed, ERwin is (said to be) able to export models to a wide variety of formats.

When you import any ERwin model you are interchanging just the abstract syntax of the model (and not completely), the concrete syntax is ignored. Hence, you are not able to display the diagram that represents your model. When you are using a big model, this is not a trivial problem. Indeed, the only models that will be correctly displayed in this kind of tools will be the proprietary model, i.e. those made with the tool itself because they are persisted in just one file that mix-up concrete with abstract syntax.

3.4.4 UML Profiles became DSLs at the Time of Implementation

Although next chapter will elaborate more on this matter, after building MIDAS-CASE prototypes we were on the position of saying that, when it comes to implementation, DSLs (again the term was not much used at that time) are a better option than UML profiles. Indeed, the most recognised model-driven methodological proposals have opted for using DSLs to *implement* their UML profiles (see UWE [205] and the works from Trujillo et al. for instance [231]).

3.4.5 Model-Transformation Language

On of the most important things we learnt developing MIDAS-CASE was the relevance and crucial role of model transformations in MDE.

All the transformations (model-to-model and model-to-code) bundled in MIDAS-CASE are encoded in the tool. More specifically, the XML parsers are in charge of these tasks. To that purpose, the parser navigates the extended UML class diagram whereas it generate an XML output. Such XML file is the target model. The code generation follows the same path, aside from the fact that this time the XML output stream has to conform to a different Schema: the one that defines the XML Schema or the WSDL metamodel.

This solution worked fine just for a while. As soon as we decided to introduce some minor modifications over the metamodels (the XML Schemas), we realised that we had to re-implement the parsers. This was a quite tedious and repetitive task. Moreover, encoding the mapping rules in the tool hides the logic of the application. The user has no idea about the development process that the tool is implementing. Even worse, he has no option to modify it in order to adapt to new business rules.

For the MDE vision to become reality, development tools should not only offer the possibility of applying predefined model transformations on demand, but should also offer a language that allows (advanced) users to define their own model transformations and then execute them on demand [320].

Finally, the use of general-purpose programming languages for model transformations coding is discarded bt the MDE community. As a matter of fact, the vast majority of model transformation languages adopt declarative approaches versus the traditional imperative approach of standard programming languages. For instance, when you code a model transformation, you need to keep track of which elements have been already mapped and to which output elements they have been mapped. The use of an imperative language to that purpose is an error prone task. In addition, it results on too verbose programs, very complex to manage [342].

In summary, model transformations cannot be sensibly written in a standard programming language, Object Oriented (OO) or otherwise. Instead, a DSL for model transformation is to be used.

3.4.6 Separation of Concerns: Modularization

The architecture of MIDAS-CASE favoured the modularization of the tool. The tool was thought as a set of coexisting modules or subsystems, each one providing with specific capabilities: deploying a UML class diagram, modelling a Web Service description and so on. The underlying idea was to encapsulate all the functionality related with each type of model in one place. That is, one module for each type of model. This way, to support a new type of model we add a new module whereas adding new capabilities to work with a specific type of model means modifying only the corresponding module.

You may consider this modular architecture as a separation of concerns, a traditional practice in Software Engineering [286, 340], to which MDE has also adhered from its origins [207]. The idea is very similar to the one we captured in MIDAS methodlogy: a layered architecture, with each layer representing one concern of the system.

Solution: M2DAT Architecture and Technical Design

Previous chapter has presented MIDAS-CASE, our first attempt to develop a tool supporting MDSD according to MIDAS methodology. However, the advent of MDE an its related technologies made us reconsider the design and specification of MIDAS-CASE in order to take advantage of the advances in the field. We want to move from an isolated stand-alone tool to an integrated framework. that bundles the most recognised tools supporting MDE tasks and is open to constant evolution. The result is M2DAT.

This Chapter presents M2DAT’s conceptual architecture and the decisions that drive M2DAT’s technical design. That is, which are the approaches and technologies adopted for support each MDE task in M2DAT and how they are used. In addition, in those tasks where several options could be sonsidered, we provide with a discussion on them in order to justify the final decision.

Indeed, MDE is still mainly a research field and it was just an incipient idea when we starterd to wotk on this thesis. Hence, building an integrated framework for MDSD implies studying the different options before making a decision on which is the approach or technology used to support each task. For instance, you may use a general-purpose language or a DSL to code model transformations. If you choose a DSL, then you may use a declarative language, or an imperative one. Besides, there are different languages based on each paradigm. Which is the one that best fit your needs?

In the following sections we will explain also this kind of decisions (both methodological and technological). First, we present M2DAT’s conceptual architecture and put forward some remarks on its technical design. Next, we present each component of M2DAT’s technical design. Finally, the last section of this chapter summarizes some guidelines on how M2DAT’s specification is to be used in order to develop a new module for M2DAT.

4.1 M2DAT Overview

Before diving into M2DAT’s specification, we would like to provide with a brief overview of the tool. We believe that having in mind a general idea will help in the understanding of the rest of this chapter. Thus, we first present M2DAT’s conceptual architecture to later introduce M2DAT’s technical design.

4.1.1 M2DAT Conceptual Architecture

M2DAT architecture, sketched in Figure 4-1 follows roughly the initial architecture of MIDAS-CASE (see section 3.2). It keeps a high level of modularization and can be described according to two orthogonal dimensions:

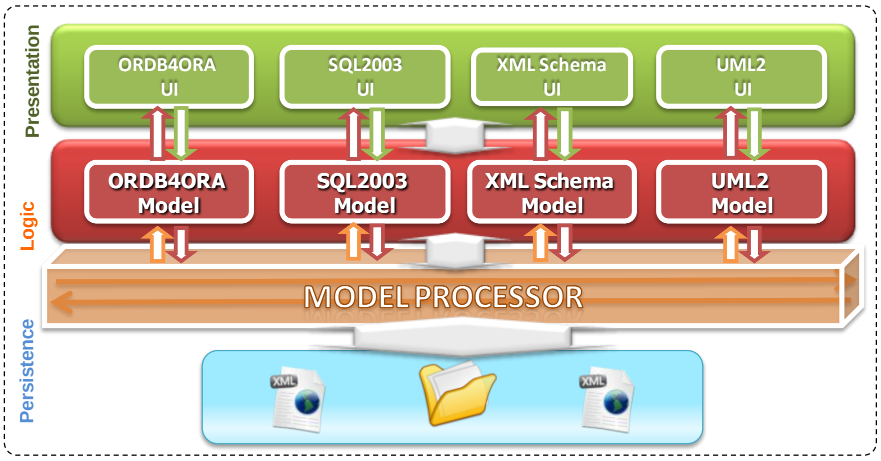


Figure 4-1. M2DAT Conceptual Architecture

On the one hand, M2DAT can be thought of as a set of modules, one for each model proposed by MIDAS to model the WIS. Each model is defined as a DSL (insights on the motivation behind this decission will be given along this chapter). So, M2DAT is a kind of workbench to work with those DSLs. To that purpose, each tool provides with the functionality needed to handle models elaborated with the DSL, like model editors or validators. For instance, Figure 4-1 shows four different modules: the one for ORDB modelling in Oracle, the one for modelling standard-compliant ORDB schemas, the one for XML Schema modelling and finally the one for UML modelling. The first three modules together constitutes M2DAT-DB, the reference implementation for M2DAT presented in this dissertation. In contrast, the last one is provided in the context of the EMP project (see section 2.1.12.1), showing the perfect integration of M2DAT with existing technologies.

On the other hand, M2DAT conceptual architecture follows the separation of concerns principle [207, 286] by distinguishing the presentation of each model from the model itself. We will show how this is semi-automatically provided by EMF and thus supported in M2DAT when introducing M2DAT technical design. This way, the **presentation** tier includes the editors (whether they are diagrammers, tree-like or textual editors) to work with each type of model supported by M2DAT while the models are handled by the logic tier.

M2DAT aims to integrate several DSLs. This implies adding support for, at least, model transformations to connect the different DSLs and you may consider also supply with model weaving capabilities. In addition, the capabilities that a DSL workbench should support consist not only in a graphical editor and code generation capabilities. The inclusion of support for model checking, model execution, etc. might be also considered [241]. We use the term model processing to refer to all these tasks, following the idea expressed in [370] to refer to all the tasks related with model handling. Indeed, the model-processing tasks constitute the **logic** of any tool for MDSD like M2DAT. Therefore, we call the module comprising all these functionalities **model processor**. For instance, when the user requests a model to be validated, it is the model processor which will carry out the validation.

Finally, the **persistence** tier of M2DAT, in contrast with the one of MIDAS-CASE, is a file system that incorporates traditional versioning policies. We have discarded the use of an XML Database since, at the moment, it just brings complexity to the development of M2DAT while most of the advantages derived from using a Database commented in section 3.2.3 are already supported (or in the way to be) by MDE technologies [30, 43, 304].

In the following we detail how this conceptual architecture is mapped into a technical design.

4.1.2 M2DAT Technical Design

After defining the conceptual architecture of M2DAT, the next step is to select the approaches and technologies to be used in order to obtain a complete specification of the tool, i.e. for each MDE task, we have to select the existing tool or component supporting such task, that best suit M2DAT's needs. To provide with a brief overview on this selection of technology, Figure 4-2 shows the main components used to deploy one particular module to support a DSL called *MyModel*. Please, note that each component and the decision to use it will be described and justified in the following sections. We might say that present sections aims at putting forward what comes.

First of all, Eclipse is the underlying platform over which M2DAT will run. In particular, all the technologies that M2DAT integrates, as well as M2DAT itself, are built on top of EMF (section 2.1.12.2 gives an overview on Eclipse architecture and EMF). Therefore, the core of M2DAT is Eclipse and the set of plug-ins that compose EMF. Later, each component or tool that is integrated into M2DAT is another plug-in (or set of plug-ins) running atop of EMF and using the models handling facilities that it provides with. Indeed, M2DAT is another set of plug-ins running atop of EMF that connects the different components that integrates M2DAT. These connections are represented by grey arrows in Figure 4-2.

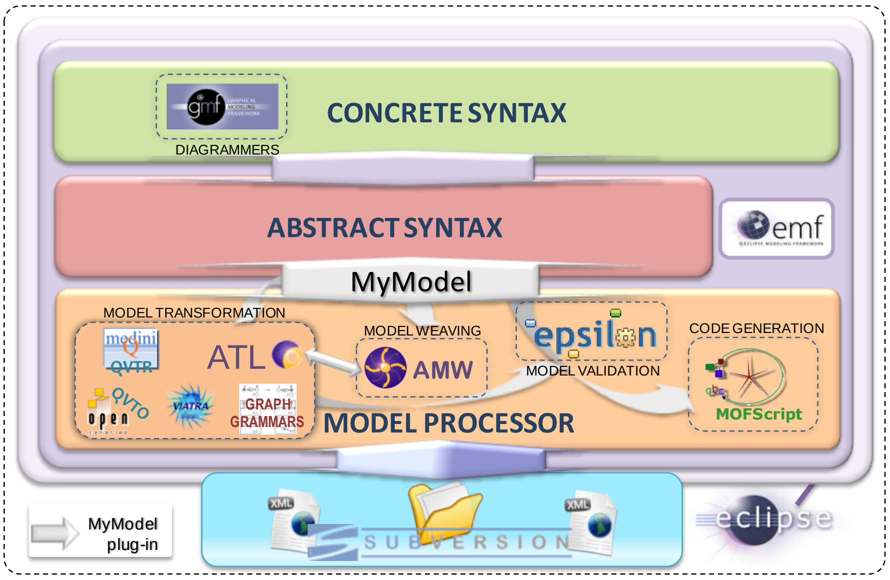


Figure 4-2. M2DAT Technical Design overview

The top of Figure 4-2 shows that the separation between the **presentation** of the model and the model itself (concrete VS abstract syntax) that was captured in the conceptual architecture is automatically supported by EMF (we will cover this matter in more detail in EMF section). To that end, EMF provides with generic editors for models plus the infrastructure to use other components, like the Generic Modelling Framework (GMF) to build more sophisticated editors.

By contrast, at the lower level we can see that the abstract syntax of the model is used as input or output in any model processing task. The support for these tasks is embedded in the **model processor**:

- The **model transformations** are developed using ATL as transformation language (see section 2.3.3.2) since it has been identified as the most convenient in order to get a reliable and efficient tool for MDSD (we will justify this statement in forthcoming sections). However, a secondary objective of M2DAT is to test the different MDE technologies. This implies things like replicating the same task, like model transformations, with different technologies in order to compare them. This way, some transformations have been also coded using QVTo from OpenCanarias (see section 2.3.3.13), mediniQVT (section 2.3.3.12) and VIATRA (section 2.3.3.10).

- In addition, we have mentioned that one of the main contributions of M2DAT is the use of **parameterized transformations** as a way towards supporting the introduction of design decisions without reducing the level of automation. To that end, M2DAT leans on the ATLAS Model Weaver (AMW) tool to define weaving models that are used as annotation models that drive the transformation. In addition, the AMW tool is also used for its original purpose, i.e. to define **weaving models** that allow defining the relationships between the elements of two models. In this sense, weaving models will be used to establish the correspondences between the models used to model the different concerns of the WIS. Those models plus the weaving models that link them are later processed by ATL model transformations.
- **Code generation** responsabilities fall on the MOFScript language (see section 2.3.4.5) so far. However, we are planning to integrate also XPand in order to compare it with MOFScript performance.
- Regarding **models validation**, the Epsilon Validation Language (EVL) is used to define constraints over metamodels that are later checked over conforming models.

Finally, we have already mentioned that the **persistence** of models leans on a traditional version control system. In particular we use another Eclipse plug-in so-called Subclipse, in order to handle the different software artifacts created with M2DAT. Subclipse is an implementation of the recognised Subversion for the Eclipse platform.

All things considered, M2DAT is a framework that integrates the best tools supporting each specific MDE task in order to obtain an efficient tool. Even more relevant is the fact that M2DAT is completely open to integrate new tools or emerging technologies that, either provides with new functionalities or just improve the tools supporting the existing ones.

We would like to mention that developing a tool like M2DAT, that integrates different tools for supporting MDE tasks, means living on an ever-lasting beta version. Even after making your decision, you should be ready for changing. New and better products might come or new releases of the underlying technology might appear. It is not our intention to threaten the reader with this discouraging outlook. The design decisions that drive M2DAT specification have been taken to be able to cope with constant evolution.

In the following we present each technological decision that drive M2DAT's technical design plus the components used to support each MDE task. Note also that, in some cases the technological decision is preceded by a methodological decision, i.e. before selecting a technology we may have to select an approach. For instance, we have to select a model transformation approach before choosing a model transformation language. Both type of decisions will be reasoned in forthcoming sections.

To that end, we follow the same structure we have followed in this dissertation so far. First, we tackle the decisions on the way to define new modelling languages. Next, the ones related with supporting graphical management of models. Then, we focus on model transformations, putting a special emphasis on model-to-model transformations. The use of annotation models follows and finally we conclude with the discussion on the integration of automatic model validation.

4.2 Modelling and Metamodelling

As clearly stated along this document, the building block in MDE are models. Any MDE proposal is based on the definition of new modelling languages. To that end, two different approaches may be followed: the traditional one, based on the extension of the UML standard in an UML profile, or the more trendy at present, based on the definition of a new modelling language (almost) from scratch (i.e. a completely new DSL).

In the following we present both approaches and provide with a discussion on them in order to justify the methodological decision taken at the time of developing the M2DAT specification: to combine the use of UML models at higher abstraction levels with the use of DSLs at lower abstraction levels.

4.2.1 UML Profiles

UML provides with its own extension mechanism in order to allow extending the language in a controlled way. Those mechanisms allow creating new building blocks by means of stereotypes, tagged values and restrictions. This way, an UML Profile is a package that contains model elements that have been customized for a specific domain or purpose using extension mechanisms, such as stereotypes, tagged definitions and constraints [70, 270]. A very common way of applying UML was to first define a UML profile for a particular problem or domain and then to use that profile instead of or in addition to general UML. This traditional way of working, let us contemplate a UML profile as a way to produce a domain-specific language (DSL) [317].

In effect, a UML 2.0 stereotype is defined as if it was simply a subclass of an existing UML metaclass, with associated attributes (representing tags for tagged values), operations, and constraints. Finally, you can also use the UML 2.0 profiling mechanism to view a complex UML model from multiple, different domain-specific perspectives—something not generally possible with DSLs. That is, you can selectively "apply" or "de-apply" any profile without affecting the underlying UML model in any way. For example, a performance engineer may choose to apply a performance modelling interpretation over a model, attaching various performance-related measures to the model's elements. An automated performance analysis tool can then use these to determine a software design's fundamental performance properties. At the same time and independent of the performance modeler, a reliability engineer might overlay a reliability-specific view on the same model to determine its overall reliability characteristics.

The language extension mechanisms were slightly restructured and simplified for a more direct way of defining UML-based domain-specific languages. These languages have the distinct advantage that they can directly take advantage of UML tools and expertise, both of which are abundantly available

In subsequent revisions of UML, the notion of a profile was defined in order to provide more structure and precision to the definition of Stereotypes and Tagged values. The UML2.0 infrastructure and superstructure specifications have carried this further, by defining it as a specific metamodelling technique. Stereotypes are specific metaclasses, tagged values are standard metaattributes, and profiles are specific kinds of packages [270].

It is worth mentioning that the first MDE methodological proposals based adopted UML profiles as modelling language. Since UML was (almost) the unique modelling language known, they opted for extending it to support the abstractions considered in their proposals. For instance, the most recognised proposals for Web Engineering, like UWE [198], MIDAS [226] or OO-H [156] were (initially) based on the use of UML profiles.

4.2.2 DSLs

In software development, a Domain-Specific Language (DSL) is a programming language or specification language dedicated to a particular problem domain, a particular problem representation technique, and/or a particular solution technique. The concept isn't new—*special-purpose programming languages* and all kinds of modelling/specification languages have always existed, but the term has become more popular due to the rise of domain-specific modelling.

The opposite is:

- a *general-purpose programming language*, such as C or Java,
- or a *general-purpose modelling language* such as the UML.

Therefore, DSLs are languages tailored to a specific application domain. They offer substantial gains in expressiveness and ease of use compared with general-purpose programming languages (GPLs) like UML in their domain of application [227]. In other words, while a DSL is designed to solve a delimited set of problems, GPLs are supposed to be useful for much more generic tasks and thus they cross multiple application domains. In other words, a GPL aims to provide with a way to represent abstractions from any particular domain. A given DSL provides means for expressing concepts derived from a well defined and well-scoped domain of interest [184]. Furthermore, the rules of the domain can be included into the language as constraints, disallowing the specification of illegal or incorrect models. Examples of DSLs range from the Structured Query Language (SQL) [176] to the SED Linux utility for matching and replacing regular expressions.

As well, it is worth mentioning the traditional difference stated between **Internal** (aka as embebbed) and **External** DSLs [213]. The former is a DSL built of constructs in a surrounding programming language. Therefore, the host language restricts and influences the syntax of the DSL. The later is built from the ground up, what means that you'll need to specify a grammar, develop a parser, etc. They seem to be small programming languages.

The trend is to design and develop and internal DSL when an organization that uses a GPL needs a technical API for specific tasks. A well-known example is Ruby on Rails [301], a Ruby-based DSL for Web applications development. In general, the ability to develop an internal DSL depends on the features of the underlying language. Dynamic languages like Smalltalk or Ruby itself results more convenient that JAVA or C#. In contrast, when you want to provide help to non-software experts or developers in the form of support for design and development tasks, you better go for an external DSL.

Another feature of external DSLs is that, opposite to GPL, they are not compiled to executable code. By contrast, they use to be translated to the language used by the underlying framework. This is the case of EMF-DSLs, where the underlying language is JAVA.

Finally, it is worth mentioning that when we refer to DSLs in this document we are always referring to external DSLs unless explicitly said.

In general, Language Workbenches (DSL development frameworks) allow you to define a new DSL abstract syntax by means of a metamodel plus a concrete syntax, typically visual. Note, however that there are some frameworks dedicated to define textual concrete syntaxes for DSLs, like xText [122], TCS [179] or TEF (http://www2.informatik.hu-berlin.de/sam/meta-tools/tef/). From those specifications, the framework provides with a graphical and/or textual editor and automatic storage of models. Later on, and depending on the framework, you may add code generation capabilities from those models and even code model transformations between your DSL and others.

As stated in chapter 2, there are several frameworks to work with DSLs, like MetaEdit+, GME, or the well-known DSL Tools from Microsoft etc. However, during the last years the vast majority of MDE research proposals (based on the use of DSLs) has adopted Eclipse and more specifically EMF as its preferred DSL framework. Mainly because using a common basis (i.e. a common underlying framework) simplifies enormously interoperability issues. The basis of DSL definition and construction in EMF is Ecore, a common metametamodel, which can be seen as a simplified or industrialized version of MOF, whose implementation is based on JAVA. We use to refer to those DSLs as MOF-based DSLs. This way, if you define the metamodel of your DSL as a model conforming to Ecore, the task of building bridges between your DSL and any other built on the EMF framework does not have to be simple, but at least feasible.

Finally, we would like to mention that there is a growing trend in the Microsoft and Eclipse communities to use the term DSL to refer to Graphical DSL, but this has not to be the case. We may define a DSL and then we may opt for adding a graphical notation or not [140, 190].

4.2.3 Discussion

The dilemma between UML Profiles and DSLs has been in the air since the beginning of MDE [375]. In fact, we may see it as part of an older dichotomy: **agile modelling** VS **monolithic modelling** [47]. The underlying idea could be summarized as: Shall we use just one big modelling language (like UML) to model the whole system or it is preferable to use a different set of abstractions to model each part of the system?

In an organization where UML has been used as modelling language UML, and that implies almost any software engineering organization, defining a new UML profile might be the quickest approach to build a graphical DSL. If your organization has been used an UML tool, the task will be done in relatively short time with relatively short effort. However, as an internal DSL is constrained by the hosting language, a UML profile is constrained by UML itself. Thus, the composition rules you may define for your DSL will have to be an extension from those defined by UML. Likewise, all the native UML information will be present in the modeller, which is distracting if it is not relevant to the domain [140].

Although at the beginning there was a huge trend towards extending UML as a way to define new DSLs, we can state that some years later UML profiles are not taking off. As a matter of fact, quite a lot of methodological proposals based on MDE were initially based on UML profiles. However, when researchers started to develop the technical support for their proposals, the above-mentioned drawbacks become more apparent. As a result, those proposals, originally based on the use of UML profiles, moved to the use of DSLs. This is the case of the already referred MIDAS or UWE. We can find even works that use UML profiles as a *formal* way of specifying their proposal, but uses MOF-based DSLs to deploy them [231].

In this sense, the use or UML profiles or MOF-based DSLs has been conditioned by the effort needed to develop the tooling support for any MDE proposal. Just to show how this fact has influenced research works on any MDE field, we now have a look at the situation of Model-Driven Web Engineering (MDWE, [200]).

When talking about CASE tool support it should be noticed that the proliferation of technologies and tools for developing "*your own*" MDE tools is facilitating the adoption and implementation of MDA principles and techniques. Many software companies and research groups are really considering the development of their own CASE tool for supporting their own MDE method (following the MDA, Software Factories, Product Lines, Generative Programming of whatever other more specific model driven proposal). This way, technology is playing a key role in the distinction between UML based and non-UML based tools: the facilities provided in the context of the *Eclipse Modelling Project* (EMP) and other DSL frameworks, like the *Generic Modelling Environment* (GME) or the *DSL Tools*, have shifted the focus from UML-based approaches to MOF-based ones. Special attention has to be paid on the EMP. The quantity and quality of the MDD facilities provided in the context of this project (a common modelling framework like EMF, meta-editors like GMF, transformation engines like ATL or

VIATRA, code generators like MOFScript) has given rise to a new generation of Eclipse tools. As a consequence, more and more MDWE proposals are developing their tools as Eclipse plug-ins, like the OOWS suite [347] and M2DAT itself, or at least, upgrading or re-defining them to be “Eclipse compliant”, like WebRatio [6] or ArgoUWE [205].

Another factor in favour of using DSL resides on the storing format. UML models are to be persisted using XMI [275], an OMG standard that aimed at making reality the never-kept promises of UML advantages in the form of interoperability. Unfortunately, XMI for UML has turned out as an additional complication because mainly of its versioning problems: each tool uses a different XMI version, thus UML models are not exportable-importable. Besides, the verbosity of XMI complicates enormously handling UML-XMI artefacts. In contrast, when you develop your own DSL you can define your own XML format for models storage.

Even Microsoft, always far away from the OMG standards, is adopting such approach: they refer to it as **pragmatic modelling** and is based on the combination of UML and DSLs [321].

Finally, the ability of applying or de-applying UML profiles in order to have different views of a same model is compensated with the use of different DSLs to model the different views of the system plus the use of weaving models to weave those views. The result is much more flexible than using an unique model to specify the whole system.

Regarding DSLs, some authors claim that, since those languages are closer to the problem domain than to the implementation domain and follows the domain abstractions and semantics, they allow modelers to perceive themselves as working directly with domain concepts [299]. However, in our opinion a DSL is not just valid but the best option to model also the solution domain. In fact, we follow a well-known principle, followed by some of the most important Software Engineering practitioners [136]: we use UML to model the problem domain (for instance Conceptual Data models or Use Case models). UML is recognised as the best language to for analyzing and designing the architecture of the enterprise. It is more intuitive and thus more convenient to be used when transmitting ideas to the business architects and the like. On the contrary, we use DSLs to model the solution domain, like the ORDB model, the XML Schema model or the WSDL model. When we talk about the solution domain, we are referring to IT/SW issues. In this case, we need from more detailed and specialised models if we really aim to generate working code. Thus, UML is too generic for these tasks. Moreover, the stakeholders in this case will be software developers, not very prone to the use of UML.

All things considered, our response to the dichotomy between UML-profiles or (MOF-based) DSLs is not just A or B. We bet for a mix of both approaches. We use **pure UML at the higher abstraction levels** (like Class Diagrams or Use Case models) and **DSLs at the lower abstraction levels**, when we are close to the final platform and thus we need models that are more detailed. However, we do not discard UML completely at those lower levels, since we try to define UML-like DSLs. That is, although the underlying language will be an Ecore-based DSL, the concrete (visual) syntax will be that of UML. This way, we define our own metamodel for Activity Diagrams, Use Case models, etc. Working this way, we take advantage from the main contribution of UML: any Software Engineer is capable of recognising a UML model. At the same time, we get rid of the main drawback of UML: the size of the specification. For instance, when you are defining a Use Case model, where is the sense of having to navigate the whole UML specification whenever you want to check whether a particular property exists or not? This kind of situations has a dramatically input on performance when you are using the models as input for code generation or model transformation tasks. Your model parser need to navigate the whole UML metamodel, while just a little part is needed for defining the different models for the different parts of the system.

4.2.4 Selecting a Metamodelling Framework: EMF

Previous section has focused in presenting and justifying a methodological decision: the selected approach for modelling languages in M2DAT specification. The logical step that follows is to translate such methodological decision into a technical decision, i.e. it is time to choose the technology to develop M2DAT's DSLs. In the following we justify our decision on this matter and our bet for EMF as metamodelling framework.

4.2.4.1 Combining DSLs with UML Modelling

According to the methodological decision that states that M2DAT has to combine UML modelling with the definition of new DSLs, the first requisite of the selected technology is to **support also UML modelling**. Indeed, the movement from high abstraction levels to low abstraction levels, i.e. from UML models to DSL models, will be carried out by means of model transformations and weaving models. Hence, the better the integration between UML and the DSLs used, the easier it will be to make this movement downwards.

This requisite inhabilitates almost all the metamodelling frameworks reviewed in section 2.2. In fact, all of them were focused on the definition of new DSLs but provide with no support for UML modelling. Besides, their closed and isolated nature hampers the building of bridges to bring UML models to such frameworks. By contrast, EMF is a DSL toolkit [161] that supports UML modelling since the EMP includes the UML2 sub-project, an EMF-based implementation of the UML2 standard [270].

This way, the fact that the new DSLs developed in the context of M2DAT will be defined over the same metametamodel that UML (at least, a widely adopted UML implementation) eases the task of bridging UML models with DSL models.

4.2.4.2 Interoperability

We have already mentioned that a recurrent problem regarding tool-support for MDE is **interoperability**. The main explanation to this issue lies in another recurrent problem in software engineering: the gap between standards and their implementations [34, 132]. For instance, we have shown how the advantages that XMI was to bring as format for models interchange has never come to reality because of the different interpretations of the standard that each manufacturer has done at the time of implementing it.

In order to solve this gap, software developers tend to agree in a common implementation close enough to the standard and adopt it as reference implementation. In fact, the current trend towards the use of EMF has resulted in a wide community of EMF users and developers. The most outstanding research organizations in the field of MDE are developing their prototypes using EMF. As a result, the use of EMF as metamodelling framework leverages the level of interoperability of M2DAT since M2DAT’s models could be imported/exported from/to the most recognised and accepted tools in the field of MDE.

As a matter of fact, the use of EMF as metamodelling framework eases the task of finding the right tool to support the rest of MDE tasks. Since the most adopted and mature tools for MDE tasks have been also developed in top of EMF, we can use any of them without the need of an extra effort to import/export M2DAT’s models from/to such tools. The state of the art from Chapter 2 showed that model transformations are the example par excellence of this statement.

All this given, we can conclude that the use of EMF is basic for any MDE proposal to success in current panorama. In fact, M2DAT architecture is a simplified version of MIDAS-CASE architecture, where the facilities provided by the underlying framework (EMF) solve for free some problems that were solved in

MIDAS-CASE by developing a specific component, like the XML parsers or the separation between abstract and concrete syntax of the model.

4.2.4.3 Extensibility

M2DAT aims at supporting model-driven development of WIS according to MIDAS methodology. MIDAS itself is constantly evolving and incorporating techniques to include and support the development of new concerns in the WIS. Hence, M2DAT has to be also open to incorporate support for the modelling of the new concerns as well as for connecting such models with the models of the already suported concerns. Therefore, **extensibility** was clearly identified as another mandatory feature that M2DAT had to meet.

In this context, EMF and Eclipse comes as the perfect platforms to build an extensible framework. Indeed, Eclipse is conceived as an extensible framework that provides with the basic infrastructure to be extended and was thought to that end. Likewise, EMF itself is also an open framework that is permanently evolving and incorporating emerging technologies.

Hence, we will be able to:

- Integrate any new functionality developed upon EMF on M2DAT. Just think of a new model transformation engine, a better code generation tool or whatever. As long as they are developed using EMF, they will be completely compatible with M2DAT.
- Integrate support for new model processing tasks in M2DAT. For instance, right now M2DAT does not incorporate any model comparison facility, but it will be feasible to integrate it once the tools developed in the framework of the *EMF Compare* project [65] are mature enough.
- Test new prototypes and approaches as soon as they are liberated.
- Develop and integrate new modules to support the inclussion of new concerns in the development of the WIS.

After selecting a metamodelling framework, i.e. a toolkit for defining the new DSLs that will be bundled in M2DAT, the next step is choosing the technology to be used to develop graphical editors for such DSLs.

4.3 Development of Graphical Editors

When we define a new DSL we start by defining its abstract syntax with a new metamodel. Next we need to define its concrete syntax. This means associating a notation to each concept and relationship collected in the metamodel.

Traditionally this notation has been identified with a visual shape, though it could be just a textual notation. In fact, we find the later more useful when using the DSL for code generation purposes as we will explain later. However, we should not dismiss the relevance and the utility of visual representations of models, which has been typically identified as a one of the technological foundations support for MDE [21].

In this section, after presenting the two options considered for the development of the traditional boxes and arrows model editors (aka *diagrammers*) for new DSLs in M2DAT, we justify the final decisions on this matter.

Besides, notice that using EMF as DSL toolkit we do not need to worry about the support for UML graphcial modelling since the UML2 project already mentioned (indeed, it is the UML2 Tools project) already provides support for this task.

4.3.1 JAVA Graph Components

In general, any GPL provides with some libraries for graphics that allow defining user interfaces. For instance, when we developed the MIDAS-CASE prototypes presented in the previous section, we used JAVA graphics capabilities to build the model's editors (see sections 3.2.1 and 3.3.5.1 for a more detailed insight on this issue).

However, coding a user interface from scratch is a very tedious task since you have to add the needed code not only to depict any detail of your GUI, but also to detect any user interaction and make your GUI react properly. That is, you are responsible of event handling in the diagram and reflecting the effect of the event in the underlying model.

To help on these tasks, there are several graph components available for JAVA, like JGraph [10, 11], the one we used to develop MIDAS-CASE GUI. These components provide with an additional abstraction layer over the JAVA Foundation Classes that serve to ease the development of graphical editors. They provide with abstractions to add in the GUI panel editors, boxes, arrows, widgets for properties edition, etc. Besides, the whole framework is based on the MVC pattern [148], thus it provides also with the corresponding event handlers for each widget of the GUI. In addition, those graph components provides with the traditional capabilities you would expect from a model editor, like zooming, folding, undo, drag and drop, etc.

Using these facilities we were able to develop MIDAS-CASE's model editors in a systematic way.

4.3.2 *GMF*

The Graphical Modelling Framework (GMF) [341] provides a generative component and runtime infrastructure for developing graphical editors based on EMF and GEF (Graphical Editing Framework) [250]. Figure 4-3 shows the dependencies between those Eclipse components.

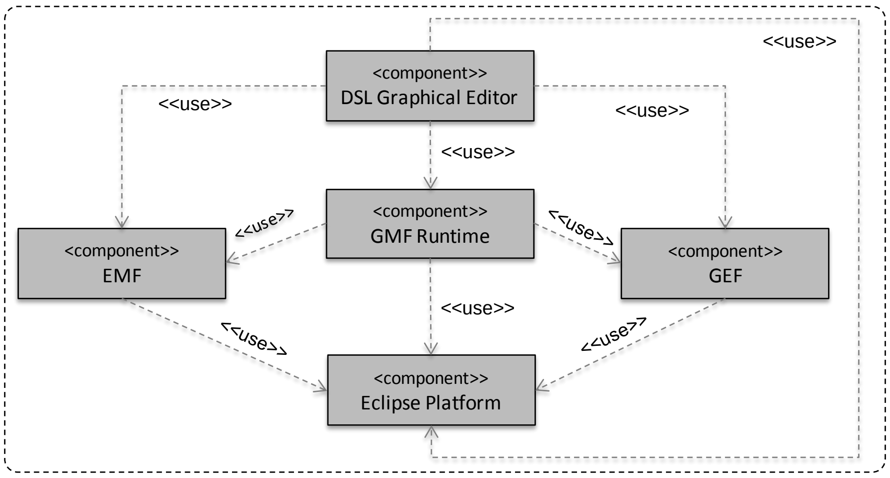


Figure 4-3. Dependencies between GMF, EMF and GEF

Any GMF editor depends on the GMF runtime and uses the EMF, GEF and Eclipse platform. Before the advent of GMF, an Eclipse model editor was developed by binding the EMF model with the GEF view by hand-coding. GMF undertakes this task replacing the coding by modelling to provide an easier way to develop graphical editors using GEF and an underlying EMF model [161].

The underlying idea is that a set of models serve to define the concrete visual syntax of the DSL and collect the correspondences between the EMF model (the abstract syntax) and the graphical elements. From such models, GMF generate the code that implements the graphical editor in the form of an Eclipse plug-in. The development process is depicted on Figure 4-4, detailing the different models that you should define to build a GMF editor.

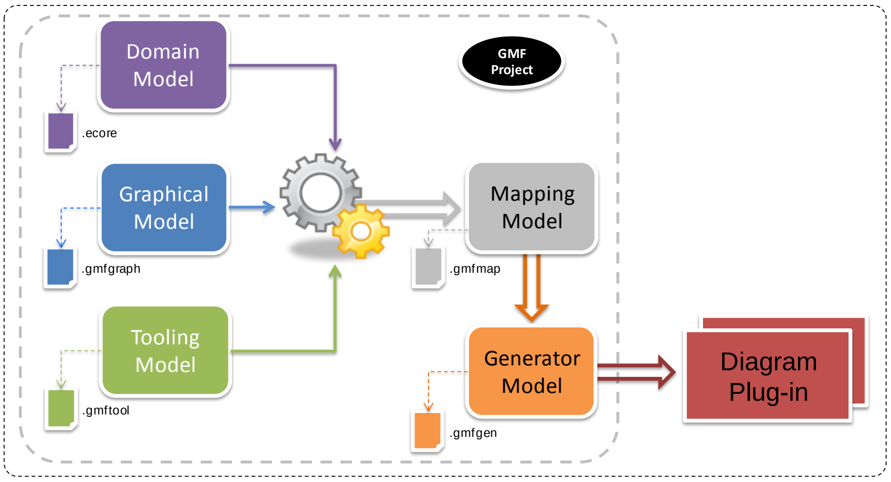


Figure 4-4. GMF Development Process Overview

- The **domain model**: this is the Ecore metamodel used to define the abstract syntax of the given DSL. It defines the non-graphical information managed by the editor.
- The **graphical definition model**: this model defines the graphical elements to be displayed in the editor.
- The **tooling definition model**: it states which are the widgets that compose the user interface of the editor. In essence, it defines the tool palette.
- The **mapping model**: finally, this model links the previous models together. Graphical and tooling elements are linked with their corresponding elements form the domain model. In other works, it bridges the abstract syntax of the DSL with the concrete (visual) syntax plus the widgets to add each different modelling element to the diagram.

GMF tries to simplify the tasks of defining these models by providing with wizards that drive the user on the process to define each one. In addition, a tentative mapping model is automatically generated. It is a first attempt to match the domain, graphical and tooling model. From that initial mapping, the user has the the right to modify the mappings identified as needed.

Once the above models have been defined, GMF generates a new model, so-called the generator model. This model encodes implementation details that will drive the generation of the final plug-in that implements the diagrammer.

This way, the main features of GMF are **reutilization** of the graphical definition for different domains and applications and **automatic** generation of the diagrammer**.**

On the one hand, since the only connection between the domain concepts and its graphical representation is the mapping model, we just have to modify the mapping model to reuse the graphical abstractions already defined for any other domain. On the other hand, GMF applies MDE techniques. The diagrammer is automatically generated from a set of models applying model transformations. Actually, until recently they were not proper model transformations since JET was used to generate the diagrammer code. More recently, Xpand has been adopted to support this task. All this given, GMF is a perfect example of MDSD.

Finally, if you are ok with the default capabilities of a GMF editor, you do not need to touch a single line of code since the whole process is automatic. However, you still have the right to modify the generated code to obtain a different look and feel for your editor or to add/modify the capabilities provided by GMF.

4.3.3 Selecting a Technology to Develop Model Editors

In the following we expose the main reasons in order to select GMF as the technology to use in order to develop diagrammers for a DSL plus our bet for EMF tree-like improved editors as default editors for M2DAT's DSLs.

4.3.3.1 Compromise between development effort and result

In general, JAVA Graph components are much more powerful than GMF as a tool for graphical editors development. The former provide with a higher level of control and detail over the final result. In contrast, a GMF generated editor can only be customized to some extent in reasonable time and manner. In addition, the look and feel of a GMF editor does not always fit the user needs. For instance, Figure 4-5 shows that anchoring between shape is not very accurate. Notice the space left between the connector and the node to anchor.

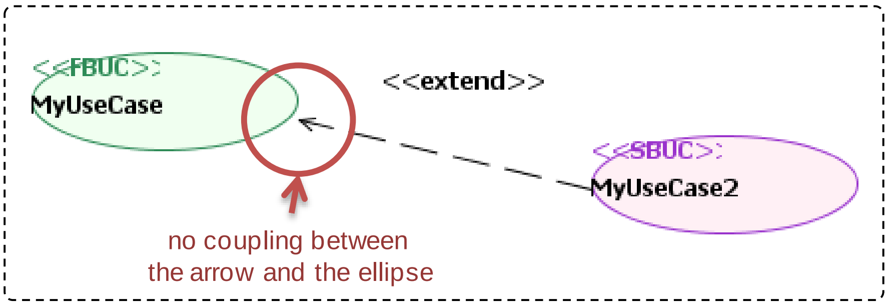


Figure 4-5. GMF anchoring problems

Although this issue could be addressed by modifying GMF generated code, it is a very challenging task. The problem lies in GMF internals. GMF's developers had to compromise the simplicity of the architecture for the sake of genericity. However, the advantages derived from using GMF are still compensating this drawback.

4.3.3.2 Interoperability

When we argued in favour of using EMF as metamodelling framework (see section 4.2.4), we already mentioned the advantages provided by EMF in terms of interoperability.

Being developed atop of EMF, GMF shares its beneits in terms of interoperability. Hence, while JAVA Graph components and some other exiting metamodelling frameworks, like MetaEdit+ (section 2.2.10) offers more accuracy when developing graphical editors development, the underlying modelling framework of GMF (EMF) provides with direct interoperability with a wide variety of MDE tools.

To sum up, models elaborated with a GMF diagrammer preserve the advantages of EMF in terms iof interoperability. In the case of M2DAT, this fact is a winning argument in favour of selecting GMF for model editors development.

4.3.3.3 On the relative relevance of diagrammers in MDSD

Finally, we would like to downplay the relevance of diagrammers in MDSD. As a matter of fact, though graphical editors are an useful and important component in any MDSD tool, we have realised that in some scenarios they are not the best tool.

When we first addressed the development and construction of first M2DAT's prototypes we thought that diagrammers were an essential piece of the tooling to be developed. Indeed, they had been traditionally one of the most commonly used and accepted tool for Software Engineering tasks. Actually, diagrammers were essential when models were nothing but additional documentation for the projects. Then, those diagrams offered a good overview on the domain analysis, the system design, the business process and so on. However, in the context of MDSD projects, a model became software itself. Hence, a brief overview of the model is not enough. For instance, think of JAVA/C# framework providing a capability for simultaneous editing in textual and graphical mode. It is assumed that the graphical view provides a limited view on programs [99].

If we want models to be mapped directly to working code, we need extremely detailed models. A graphical editor is not always capable of providing with the level of detail demanded to that end. Note also that our approach to modelling languages, where DSLs are used at lower abstraction levels while UML is used at higher abstraction levels, implies that M2DAT's DSLs are close to deployment platforms. Hence, the need of accurate editors is a must for M2DAT.

In this context, during the construction of first M2DAT's prototypes we found that the simple tree-like editors provided by EMF resulted much more convenient to work with DSLs modelling PSMs that are directly translated into code. As a result, we have worked in identyfing the mechanisms and techniques to improve such basic editors and customize them to the needs of each particular DSL. Some of them will be introduced in section 5.2.2 when presenting the reference implementation for M2DAT.

However, we still think that diagrammers are needed in any MDSD tool for providing with first sights of any given model. Thus M2DAT's specification states the toolkit for DSL supported by M2DAT has to bundle a diagrammer for conforming models, though the effort dedicated to its development should be considered very carefully. In this context, the generative nature of GMF fits perfectly with our purposes regarding graphical editors development: GMF generates an efficient though not perfect diagrammer in reasonable time and manner.

4.4 Model Transformations: the Kernel of a MDSD process

Model transformations are the masterpiece to drive any MDSD proposal forward given that each step of the development process involves a model transformation to create or generate a new model from one or more input models [41, 316]. This way, once the DSLs of the proposal have been defined and the toolkits to work with them have been developed, the next step is to bridge them by means of model transformations [246].

As a matter of fact, without automating the mappings between models, the effort needed to manually transform the models become prohibitive and organizations will not get a full return on MDE's promise of faster, less costly software development [134].

The state of the art provided a complete study on the exiting solutions and approaches to develop model transformations. Hence, this section will focus just on explaining theselection of approaches and technology made regarding how model transformations are to be developed in M2DAT. To that end, the following sections revisit some of the conclusions already sketched in section 2.3.

Given the relevance of model transformations, we separate this discussion on the following points: discussion about the convenience of using a GPL or a DSL for developing model transformations; selection of a model-to-model transformation approach; selection of a (-n hybrid) model transformation language and finally, some comments on the comparison between the seleted language and existing implementations of the QVT standard.

4.4.1 GPLs vs DSLs

The first decision to make is to choose the generic way of address the development of model transformations: we might use a GPL or a DSL for model transformation. We opt for using a **DSL approach**.

The use of a DSL allows defining model transformations as transformations models [49] and thus allows model-driven development of model transformations [349]. Working with transformation models provides with several advantages:

- During early stages of the development process, it might be preferable to concentrate on the properties of the transformation by collecting them in a transformation model, that on how it is implemented.
- We can handle and produce transformation models using the already mentioned Higher Order Transformations (HOT), that is, transformations that consume and/or produce transformation models [350].
- Besides, we can use refactoring or composition techniques to build new transformation models [51, 309].
- As any other type of model, transformation models can be validated and checked with existing tools [72, 218].

- Finally, if we are able to identify a common metametamodel for model transformation languages, we can migrate model transformations expressed in a particular language to any of the others transformation languages conforming to such metametamodel.

Likewise, the DSL approach is the recommended way of working by the OMG itself. In fact, their proposal for model transformations development, QVT, is nothing but a recommendation/normative to follow in order to build a DSL for model transformations.

Finally, the use of a DSL eases the development of model transformations. Since the language is focused in providing support for an specific task, it typically bundles some facilities that, using a GPL, would have to be implemented by the developer. In contrast, a DSL for model transformation is specifically intended to define how a set of source models have to be visited to create a set of target models. For instance, a relatively simple model transformation coded with a GPL has to add increasing amounts of machinery to keep track of which elements have already been transformed, while DSLs for model transformation use to include built-in support for this task [342].

To conclude, our first decision on how to develop model transformations is to use a DSL for model transformations.

4.4.2 Selecting a Model-to-Model Transformation Approach: the Hybrid Approach

Once we have made a bet for using a DSL, we need to state which is the preferred approach for developing transformations in M2DAT. In the following, we briefly present the reasons to discard some of the approaches identified in section 2.3.2 to later provide with a more detailed discussion on the remaining approaches.

4.4.2.1 Discarding less commonly adopted approaches

First of all, **direct model manipulation** approaches suffers from the same drawbacks already mentioned about GPLs: they were not intended for direct model manipulation. Thus, a model transformation expressed following such approach results complex and too verbose.

Next, while **XML-based** approaches work fine for transforming documents expressed with markup languages, they are not usable for model transformations where XML is used just as storage format but the resulting XML documents are far from being intuitive and easily-to-use.

While **template-based** approaches are widely used for code generation (indeed, the OMG's MOF2T standard is a template language [266]), they result too rigid for model-to-model transformations. For sinatcne, they provide very limited capability to compose patterns.

4.4.2.2 Discarding Graph-Based approaches

Due to the fact that there exists several model transformation languages following the graph-based approach, this section aims at exposing its main features and main issues in order to adopt or discard a graph-based approach.

Graph-based transformations are probably more appealing from a purely researcher point of view. Graph grammars are based on a solid mathematical theory and therefore they present a number of attractive theoretical properties that allows formalizing model transformations. In addition, we can think on (visual) models as graphs. A graph has nodes and arcs, while a model have classes and associations between those classes; this way the fact that models are well represented as graphs is particularly appealing to shorten the distance between modellers and model transformation developers, a big problem around model transformation. Rule-based transformations with a visual notation may close the semantic gap between the user's perspective of the model and the implementation of transformations [377].

However, the level of formalization brought by graph-based transformations does not make up for the complexity added to the development of transformations. Expressing a model transformation in terms of visual graph-rewriting rules is too challenging. As a matter of fact, existing languages use to need from textual constructions to be able to define the transformation. Even in some cases, like VIATRA, the visual representation is not supported, thus one of the main advantages of graph-based approaches broke up.

In addition, it is worth mentioning that their level of adoption is rather low when compared with DSLs for model transformation. In general, their use is limited to the teams that develop them, that use to publish works showing where their language is successfully applied to solve some SE problem. We believe that this issue is directly related with the inherent complexity of graph-based transformations. This complexity hampers the adoption of a tool developed by others, since you have to learn, not only how the mappings between your metamodels are defined with graph rewriting rules, but also which type of rewriting rules are used in the particular language (each language uses different notations), how the rules are sequenced, etc.

In contrast, we have experienced that DSLs for model transformations are much more similar. Once the mapping rules to code have been clearly identified, expressing them with different languages is a feasible challenge (given that they follow a similar approach).

Finally, it is worth mentioning that, although not for transformations, graph-based approaches have been widely accepted for the rest of model processing tasks, especially for simulation of models with dynamic features. As well, they are still valid for specification purposes, though a graph-based specification has to be later compiled into operational mechanisms.

4.4.2.3 Discarding purely Declarative and Imperative approaches

So far we have already discarded almost every approach for model transformation development identified in section 2.3.2. From such list, just **declarative**, **imperative** and **hybrid** approaches remain to be considered. This section summarizes our main conclusions around these approaches and states which is the final decission. To that end, it mainly focuses on comparing declarative vs imperative approaches. The former is based on defining the relations that must be kept between the input and output artefacts while the latter is based on explicit creation of target elements using a procedural style plus typical programming constructions.

Declarative languages own an implicit nature. For instance, the pattern matching mechanisms are implicit, thus there is no need to implement them in the code of the transformation. By contrast, imperative languages force the developer to make everything explicit. Hence, a transformation expressed with a declarative language use to be more concise than the equivalent imperative specification. On the other hand, conciseness might hamper understanding. Indeed, many issues remain hidden to non-experts developers in a declarative transformation since it is les explicit than an imperative one. Therefore, the **learning curve for declarative languages use to be longer**.

A major advantage of a pure declarative approach is that **each rule is completely independent** from the others. That is to say, you do not have to worry about how X elements are mapped when defining the rule to map Z elements. This way, once you master the technique of declarative programming, using a declarative language simplifies enormously the task of coding the transformation.

Likewise, imperative approaches do not maintain intermediate structures (so-called transient links). This might adds complexity to include built-in support for **traceability management** in the transformation languages.

In addition, declarative approaches implies **syntactic separation between source and target constructions**. A mapping rule in a declarative language consists of clearly distinguished source and target patterns what helps on identyfing to which model belongs a referred element in the code. In contrast, in an imperative transformation you find elements from both source and target models mixed on the code.

Regarding **rules scheduling**, there are also a remarkable difference. The execution of declarative transformations (with and appropriate transient tracing mechanism, as we will explain in section 5.3.4.2) is deterministic. Thus, there is no need to worry about this issue. In contrast, imperative style implies that scheduling of the rules is explicit and it is a mandatory task for the developer. With complex metamodels, this becomes a quite challenging task.

Finally, imperative approaches hamper (if not prevent from) defining **updatable transformations** to support change propagation. Since they focus just on how elements are to be created in the target model, without taking into account the relations that must hold between source and target elements. For example, multidirectionality or target incrementality is only feasible in the context of a declarative language.

To sum up, the *imperative* style results appropriate just in simple scenarios [97]. When you are mapping a model element following the imperative style, you are forced to visit all the nested elements. Back to the classical Class to RDMS example [46], when you code the rule for mapping classes, you have to visit all the nested elements of the class, that is, its properties, methods and association ends, and invoke the mapping rules for them. If the source metamodel is complex enough, owning a high degree of nesting, the transformation gets too complicated. Besides, declarative style is more convenient to support change propagation and traceability maintenance [342]. However, the need for imperative approaches should not be diminished. Transformations with a huge structural difference between source and target metamodel needs from imperative constructions, since they own a higher expressiveness. In other words, imperative languages are mainly for quick building of models. Its nature makes them more user-friendly to developers used to work with GPLs, whereas declarative languages offer the way to tie semantically two models and are more easily maintainable.

To conclude, since declarative languages eases the task of model transformation development but imperative constructions are needed to avoid too complex transformations, we bet for a hybrid approach where declarative style takes precedence over imperative one. As a matter of fact, the state of the art revealed that this is the approach followed by the languages that are nost widely accepted.

4.4.3 Selecting a Transformation Language: the ATLAS Transformation Language

Previous sections have focused on making a methodological decision: which is the approach selected to address the development of model transformations. Finally, we decided to use a DSL for model transformation that adopts an hybrid approach where declarative style is preferred. Now, it is time to make a technological decision: we have choose one among the existing languages following the selected approach, like ATL, RubyTL or Tefkat.

We have chosen ATL because, at present, it is considered as a de-facto standard for model transformation since the OMG's QVT practical usage is hardly called into question due to its complexity and the lack of a complete implementation ready for industrial production [57]. Unfortunately, the lack of a closed specification until recently has burdened the efforts to implement the standard.

In fact, though the scenario has evolved, it is still immature. The efforts of different groups working to provide with a complete QVT implementation have revealed different problems, difficulties and ambiguities in the current specification [149]. Thus, there is no QVT reference implementation. There do exist partial implementations, both of QVT-Relational, like ikv++'s mediniQVT, and of QVT Operational Mappings, like SmartQVT or Eclipse's QVTo. However, none of them combines both approaches (declarative and imperative), in theory, one of the strengths of QVT. Moreover, they are still to be adopted by the MDE community. As a matter of fact, one can find research works that claim to use QVT for model transformations tasks but it turns out that they use QVT just for formalizing the mapping of their proposals, while ATL is effectively used to code them (see [199] and [231] for instance). Given that they have already a QVT specification, why do they move to ATL at the time of coding? Actually QVT is not the preferable option even for those that have already specified their mapping rules using QVT.

Nevertheless, to ensure that standard-compliance was not feasible, we have tested the *usable* languages of the QVT specification (since QVT-Core is more like a byte code for QVT-Relations) as part of the work carried out in the framework of this thesis. To that purpose, we have developed the very same transformation (conceptual data model to OR logical data model) with mediniQVT [174] (that implements QVT-Relations) and the QVT-Operational Mappings implementation from OpenCanarias [306]. Both of them lose when compared with ATL (see the next section). However, we still believe in standards, and specifically in QVT. In fact, it is another reason for using ATL. As mentioned before, some work has already been done in the alignment of ATL and QVT. Even better, Obeo is on the way to finish a QVT-Relations implementation [155] based on the ATL-VM [180] (see section 2.3.3.12). Therefore, we expect that an efficient ATL-QVT bridge will be available soon.

In addition, a deciding factor in favor of ATL is available documentation. In this sense, ATL is by far, the best existing model transformation engine. It provides with a complete user manual [185]; a set of introductory examples covering the basics to know when developing model transformations with ATL [13]; a zoo of metamodels defined in several formats [24] (KM3 language, Ecore, SQL, XMI, DSL Tools XML specific format, etc.); a battery of scenarios where ATL transformations have been successfully applied (in research and industrial contexts) and a very active newsgroup that helps on solving any doubt not covered in the documentation already mentioned. All these resources are available from the ATL site (http://www.eclipse.org/m2m/atl/).

Another important factor at the time of selecting ATL as model transformation technology is its good coupling with the ATLAS Model Weaver (AMW) tool. Following sections will show that we use AMW to define annotation models that are processed by ATL *parameterized* transformations. In this sense, the coupling between ATL and AMW (in fact, they wer developed by the same research group) eases the handling of annotations in the model transformation.

Finally, we would like to mention that at the beginning of 2004, when we first addressed the development of model transformations, it was still emerging as a research field. Therefore, our decision was based on a preliminary review of the few documentation existing and some initial tests. Later on, ATL has turned out to be the preferable model transformation engine by the MDE community, and as we have mentioned this has contributed decisively to constant improvement of the engine, and what is more important for us, the documentation available. In addition, during these years we have worked intensively in the development of model transformations using not only ATL but also other model transformation languages. The experiences gathered have served to confirm that our initial bet for ATL was completely correct.

4.5 Introducing Design Decisions on Model Transformations

This chapter has already presented a set of methodological and technical decisions that allow us to specify a MDSD framework. So far, we have identified the selected modelling approach, the metamodelling framework to deploy such approach, the technical solution to build graphical editors, the model transformation approach to follow in order to bridge the different modelling languages and the language to implement such model transformations.Therefore, we are able to provide with the technical support to automatize a MDSD proposal based on the use of DSLs.

Nevertheless, while we were building first M2DAT's prototypes, we realised that a completely automatic process from requirement to final deployment is not only unfeasible, but also not recommendable. Design decisions have to be introduced to drive the development process when you move from one model to the other, especially if you are moving down towards deploying platforms. There is a need of stating how abstract concepts are to be mapped to concrete software artefacts. In some sense, we need to support the introduction of design decisions in the MDSD process.

In addition, the nature of some models makes it even more difficult to automate the whole development process. For instance, business process models present considerable differences compared to structural models that raises a number of issues concerning model transformation [256, 332]. One has to be familiar with the hidden concepts in the metamodels. Resulting ambiguities on the metamodel layer have to be solved either by reasoning algorithms or user input. We need from non-uniform mappings [153] that behave different depending on the paremeters received.

In the following we discuss the different options to address this type of issues according to the complexity and performance of each possible solution. Likewise, we put forward the reasons that drive us to use the one that was finally selected.

4.5.1 Selecting an Approach to Drive Model Transformations: Annotation Models

According to the principles of MDE, a development process must provide for the highest degree of automation. In fact, once the PIM has been defined, the rest of the process should be completely automatic. In this context, the simplest solution to the kind of problems mentioned in the introduction of this section is to

use a default value for these design decisions when coding the model transformation.

For instance, back to the classical Class to RDMS example [46], you may use either one of the following mapping rules for one to one associations: a foreign key in the table for one of the classes or a distinct table containing the primary key of each of the related classes and any link attributes. Traditional way of acting is deciding for one option and encode it in the model transformation, i.e. all the associations will be mapped by means of a distinct table. But defining a one-size-fits-all model transformation in such contexts is not enough. It may occur that, in absence of a design decision stating to do so, some constructions are never generated on the target model.

It would be desirable to be able to select the most convenient option for each matched pattern in the source model. Back to the example, given a Class diagram we would like to state that the X associaton is to be mapped by means of a distinct table while the Y association is to be mapped by means of a foreign key. In a MDE context where the different steps of the development cycle should be automated by model transformations, the only way of introducing such design decisions is providing with a mechanism to parameterize model transformations.

The need of ways of driving model transformation executions was clearly identified from the dawn of MDE and MDA. Indeed, the concept of *mark* introduced in the MDA Guide [246] is direcly related with this matter: *"A mark represents a concept in the PSM, and is applied to an element of the PIM, to indicate how that element is to be transformed"* (MDA guide, pp. 22). UML profiles have been widely used as **marks to drive the execution of model transformations**.

Nevertheless, marking the model itself we are polluting the model with concepts not relevant for the domain that it represents. In section 2.4.4.2 we already mentioned the tendency of current tools for model-driven development of DB schemas towards the use of non-pure conceptual data models polluted with logical details in order to ease the mapping to a logical model. If the conceptual model is just to be used to that end, this behaviour might be acceptable. However, if the very same conceptual model has to be mapped to another logical model, the conceptual model is not valid: the logical details it contained have to be cleared out in order to recover a pure conceptual model.

Then, given that the information to drive the mapping should not be included in the model to map, one acceptable way of expressed it is in the way of **annotations** [239]. In general, models are annotated or decorated to insert information that is not defined in the metamodel. Annotation data usually is not conceptually relevant to be part of the metamodel. For example, annotations are often meta-information used for pre-processing, testing, logging, versioning, or parameterization [114, 154, 219].

Besides, MDSD must support incremental and iterative development. This means that the mappings between models must be repeatable. So, if a mapping requires some additional input apart from the source models, this information or annotations must be persistent [344]. In a MDE context, everything should take the shape of a model. Therefore, we propose to collect this extra data or annotations in another model, so-called **annotation model**, that is attached to the source model following the decorator design pattern [148].

For instance, suppose we have a source and a target metamodel, a terminal model conforming to the former and the corresponding model transformation. Then, for each annotation model used to execute the transformation, different target models will be generated without any modification in the source model. This is the approach followed to develop model transformations in M2DAT.

The use of annotation models to drive mappings execution leverages the degree of automation in the MDSD process embedded in M2DAT. As long as the models handled are complex enough (and this use to be the case when working in real projects), complete automatization of the development process is not feasible. However, using weaving/annotation models we are providing both with a way to semi-automate the introduction of design decisions plus a way to persist them. since the annotations or design decisions will be collected in the weaving models. They can be modified and the target models regenerated to reflect the result of these modifications. This contrasts with the approach followed by other implemented proposals like UWE, where *manual refinement* tasks are to be made after the transformations of some steps of the development process have been executed [205].

4.5.2 Selecting a Technology to Create Annotation models: AMW

We have just introduced the selected approach to drive the execution of model transformations in M2DAT. Next step is to translate such methological decision into a technical decision, i.e. we have to identify a technology to create the annotation models used in M2DAT.

First, we have to consider that the need for defining an annotation model for every source model might hamper the modelling task. If the process of defining annotation models is not intuitive and user-friendly, the modelling task might result too tedious. To help overcoming this issue, instead of using or defining a completely new metamodel to create annotation models, we bet for using a weaving model (see section 2.1.7).

Weaving models are a special kind of model used to establish and handle the links between models elements. Hence, a weaving model is intended to be attached to some other models by nature. Therefore, it fits better to our purposes: each element of the weaving model will express some information about an elment from the model we want to annotate (so-called *woven* model). This way, a simple scenario of using a weaving model as annotation model is shown in Figure 4-6.

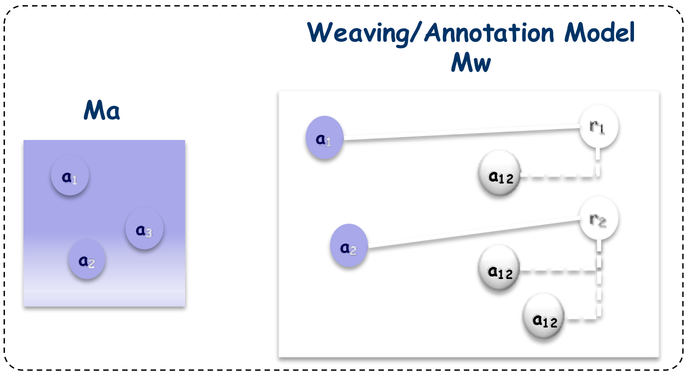


Figure 4-6. Weaving models as annotation models

Note that this scenario differs from typical scenarios where two models are woven. In this case only one model is woven: the annotated model (Ma). Both a_1 and a_2 are elements from Ma. They are annotated by linking them with the r_1 and r_2 annotations. In turn, r_1 contains a property (a_{12}) that gives *extra* information about a_1 whereas r_2 contains two properties (a_{21} and a_{22}) playing the same role with regard to a_2.

To create and handle the weaving models used in M2DAT we use the ATLAS Model Weaver (AMW). The AMW workbench provides a set of standard facilities for the management of weaving models and metamodels [114]. Moreover, it supports an extension mechanism based on a Core Weaving Metamodel that contains a set of abstract classes to represent information about links between model elements [115]. Typically, the classes from the core Weaving Metamodel are extended to define new weaving metamodels for specific contexts. One of those extensions allows the definition of annotation models and was presented also in [115]. Therefore, we could use the afore-mentioned annotation metamodel directly or use the extension mechanism supported by AMw to define new annotation metamodels for each particular scenary.

In addition, AMW provides with a GUI that adapts to any weaving metamodel extension. The user interface is automatic generated according to the metamodel extensions by using effectively the reflective API of EMF. In other words, using AMW there is no need to develop a graphical editor for annotation models. If the annotation metamodel is based on the Core Weaving metamodel, AMW generates automatically an easy-to-use and intuitive editor for conforming models. For instance, Figure 4-7 shows a screen capture from AMW.

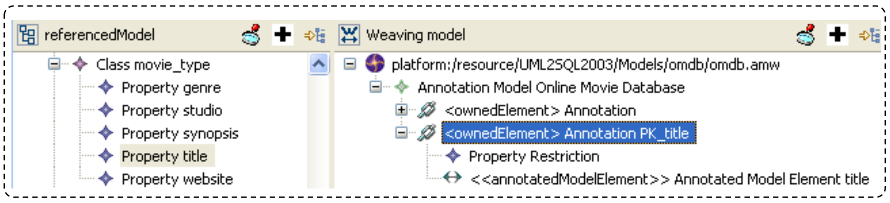

referencedModel
Class movie_type
Property genre
Property studio
Property synopsis
Property title
Property website
Weaving model
platform:/resource/UML2SQL2003/Models/omdb/omdb.amw
Annotation Model Online Movie Database
<ownedElement> Annotation
<ownedElement> Annotation PK_title
Property Restriction
<<annotatedModelElement>> Annotated Model Element title

Figure 4-7. AMW GUI Screen Capture

The panel on the left-hand side shows a UML class diagram (represented in the EMF tree-like editor) while the panel on the right-hand side shows the corresponding annotation model. Note that when the user clicks over an element from the weaving (annotation) model (the *PK_title* annotation object), the referenced element is automatically shadowed (the *title* property) and viceversa. Note also that this way of displaying a model and the corresponding weaving (annotation) model results very intuitive: at one side the reference model and at the other side, the references. Likewise, the use of AMW's GUI is quite simple. Just by dropping an element from the left panel to the right panel, AMW creates an annotation object for the selected element.

Finally, we have already mentioned that a decisive factor in favour of ATL is its good coupling with ATL. Hence, it is also a decisive factor to choose AMW as tool for creating weaving models in the context of M2DAT. Chapter 5 will show that AMW annotations are easily handled in ATL transformations.

4.6 Code Generation: the last step in the MDSD process

Any MDSD process culminates in the obtention of the working-code that implements the software system. Hence, after having identyfing the approaches and technologies to build the DSLs fo M2DAT and bridge them by means of (parameterized) model transformations, it is time to select the approach to follow for code generation and the preferred technology.

In the following we provide with a brief discussion on existing approaches and present the main motivation behind the use of MOFScript to deploy code generation in M2DAT.

4.6.1 Selecting a Code Generation Approach

The term code generation has been traditionally related with the last phase of a compiler, where an Abstract Syntax Tree (AST) was translated to source code in the targeting programming language [140]. Hence, when we addressed the specification of how code generation has to be tackled in M2DAT, code generation tasks were mainly related with **stand-alone parsers** following a template-based approach, like Velocity, Smarty, Contemplate, Cheetah, Jinja, Savant, or Liquid to name only a few.

With the advent of MDE the role of code generation gained attention. In essence any MDSD process is a chain of model to model transformations that generates models with a lower abstaction level until a model close enough to the targetted platform is obtained. Then a code generation step serializes such model into the source code. Since the input for the code generation is a model, a new term was coined to refer to code generation in MDE contexts: model-to-text transformations. As a result, a number of **DSLs for model-to-text transformation** has appeared during the last years.

Although deep down, stand-alone parsers and DSLs for model-to-text transformation are similar, the main difference lies in which is the artefact that drives the generation process.

In the former, the grammar of the language drives the generation process, the parser navigates the input programs to find matches of grammar constructions. In the latter, it is the metamodel of the DSL the one that drives the generation process. The generator navigates the input models trying to match their elements with the patterns (defined in terms of the metamodel) collected in the transformation specification. This way, in a code generation process supported by a model-to-text transformation language, the model plays the role of the AST. In fact, the classes that are instantiated when defining an AST corresponds to the classes defined in the metamodel of the DSL. In addition, we can match the syntactic sugar [213, 214] of programming languages with the concrete syntax of today’s DSLs.

The idea behind code generation remains valid, the innovation is in the way the processing is carried out. In GPL compiling, a parser walks the AST while the correspondent code is written to an output stream. In MDE contexts the approach is the same, it differs only in that the generator visits the internal representation of the model to generate the output stream.

Indeed, the former is comprised in the latter that also comprises traditional simple text replacement approaches, whose most representative example is XSLT [389]. At best, this type of generators provides with the same capabilities than a model-to-text language but a higher cost in terms of complexity and verbosity [192]. This is mainly due to the fact they do not take advantage from the metamodel to navigate terminal models. By constrast, model-to-text languages lean on the metamodel to simplify code generation. Indeed, they do not need to perform a lexical analysis, a preprocessing and a parsing phases to build the Abstract Syntax Tree (AST). They lean on the metamodel, that defines the classes of the AST to build the AST. This way, the metamodel results much more useful than a BNF grammar [140] and the task of defining the transformation is much more simpler.

Besides, DSLs for model-to-text transformations use to be metamodel-based text-generation tools. That is, they may be expressed as models conforming to an underlying metamodel. Therefore, as models, they are suitable to be used in the context of any model processing task: they might be transformed, validated, simulated, etc. In sumamry, using a DSL for model-to-text trasnformations we are taking advantage of the same issues already sketched in section 4.4.1 when comparing GPLs vs DSLs for model-to-model transformation.

Therefore, code generation tasks in M2DAT will be developed using model-to-text transformation languages.

4.6.2 Selecting a Model-to-Text Transformation Language: the MOFScript language

Afetr deciding on using a model-to-text transformation language for code generation, we have to state which is the language to use. Please, note that we stick to code generators in the EMF framework, since we have already made a decision on which the underlying framework of M2DAT is. However, this decision in not restrictive at all, since the most important (open-source) solutions are built upon EMF as the state of the art in section 2.3.4 showed. Hence, such section provided also with a brief overview on generating technologies in the Eclipse framework.

Up to now, MOFScript has been the model-to-text transformation language for generation tasks in existing M2DAT prototypes because of several reasons.

MOFScript was one of the first submissions in response to OMG MOF Model to Text RFP process [267], thus when we addressed this task it was probably the most contrasted and the most commonly used. Besides, the adoption of the visitor-based approach (very similar to traditional programming) shorten the training period. Furthermore, the visitor-based approach proven to work fine for the firsts generation tasks we tackled in M2DAT protoypes: mainly SQL code.

Before making the decision, we did some tests with the other generative technology that existed in the context of Eclipse: JET. We discarded it because of its verbosity. We find it too complex to code M2T transformations with JET since it was too JAVA-based and was not devised to work with models (i.e. it sifferes from the drawbacks already commented in previous section about standa-lone parsers).

Likewise, by the time we started to develop code generation scripts, nor Xpand, neither Acceleo MTL had appeared. Not even the OMG had delivered the final specification of the standard. Besides, when Xpand appeared it seemed to be too tightened to its underlying framework, OpenArchitectureWare. The definitions of M2T transformations with Xpand imposes the use of its workflow component.

However, right now we are revisiting those technologies, which are much more mature than they were a couple of years ago. In addition, when we have tackled model-to-text transformations for new concerns we have realised that, in some cases, template-based approach fits better and eases the task. For instance, when the code to generate is expressed in some mark-up language, like XML or HTML, the template based approach simplifies the task. Otherwise, auxiliary functions have to be coded and invoked all along the transformation program to generate repetitive constructions.

As well, it should be mentioned that so far there has been much more activity around model-to-model than on model-to-text transformation languages. Hence, it is still mainly a research field, as the late arrival of the standard confirms, where new proposals appear each day.

Therefore, we do not discard changing our preferences on code generators on the mid-time. As a matter of fact, this is one of the advantages of M2DAT. We can use or integrate new technologies in the platform as long as they are based on EMF. And on current MDE context, this requirement is met by 99% of new technological proposals.

4.7 Model Validation

The new role of models in MDE also influences the relevance of having at one’s dispossal model validation mechanisms [91]. Before MDE, models were used just for documentation purposes. Thus, their level of accuracy was not a cornerstone issue. By contrast, in MDE proposals models are the driving force. Any error in a particular model will be transmitted through the different models generated until the working-code. Model validation mechanisms can be used to detect errors and inconsistencies in the early stages of development and can help to increase the quality of the models built as well as the code generated from them. These activities are especially important in proposals aligned with MDE because it proposes that models were used as a mechanism to carry out the whole software development process [249].

We have already mentioned that the metamodel of a DSL is not enough to reach a precise and rigorous specification of which will be the valid models. The metamodel just collect the static semantics of the language, whereas some constraints have to be to defined to collect some domain rules that were not able to be collected in the metamodel [125]. Such constraints are defined at metamodel level and evaluated over conforming models to check if the model is valid. In fact, the model will be passed as input to a model transformation designed to work with correct models (whether it is a model-to-model or a model-to-text transformation). Thus, it has to be free of errors before being used by the transformation.

In the case of M2DAT specification, the need for model validation mechanisms arose when we started to build the firsts prototypes. Hence at that moment, the issue has been already tackled in MDE contexts. Therefore our decision on this matter was clearly influenced by existing works in the area. This is another point to show how the reference implementation of M2DAT influences its conceptual architecture and technical design, as Figure 1-4 showed in the Introduction Chapter.

Next two sections summarize our main findings regarding how model validation mechanisms can be supported and the reasons behind our final decissions on how it will be done in M2DAT.

4.7.1 Selecting a Model Validation Approach

There are not many approaches to implement model validation mechanisms. In essence, we can distinguish between **hard-coding the validation**

rules in the model editors provided with the DSL toolkit or defining them outside of the editors using a **DSL for constraints definition**.

Following the same reasonings spread over this dissertation, the latter approach results more convenient. Hard-coding the constraints to check in the editors goes completely against the traditional principles of modularization [116] and separation of concerns [286] since we would be mixing the code for visualizing models with the code for validating them in on single place.

Besides, working that way the constraints remains hidden from the developer, thus he would find harder to understand which were the errors he introduced that made the validation failed.

By contrast, coding the constraints outside of the editors improves modularization and extensibility. For instance, if the domain rules changes new constraints have to be implemented to define what is a valid model. Then, the toolkit for the DSL has to be updated to support the new constraints. However, working this way, it is just the constrainst checking component what has to be updated and the place where such modifications have to be made is much more localized.

Moreover, in contrast with with model-to-model transformation, code generation, model-to-text transformation or design decisions introduction, there has been a consensus about how model validation is to be implemented in MDE proposals and OCL [268] has been commonly accepted as the language to express constraints in metamodelling frameworks. Indeed, it was devised for this task from the beginning and had been already used to that purpose before the advent of MDE.

Therefore, in this section we will not go deep into the discussion around the approach followed and we will adopt one based on the definition of OCL constraints. Hence, in the next section we to compare and justify our selection among the exiting tools or technologies to integrate OCL constraints in EMF-based DSLs.

4.7.2 Selecting a Model Validation Technology: EVL

When selecting an OCL-based implementation of a model validation mechanism we have to first consider the complexity of OCL. Though it seems to be a rather simple language, its complexity is a real shortcoming and this is one of the reasons why you find such a variety of modelling tools having their own navigation language (ATL, QVT, MTL, XPand, Acceleo, JET, etc.) though OCL was devised as a universal navigation and constraint language for MOF-based models. The most of them started from OCL and ended by defining an adapted-version of the standard.

Likewise, we can find several attempts to support OCL-based model validationa in the context of EMF.

For instance, RoclET [228] allows defining UML models and adding OCL constraints over them. In addition, refactoring of constraint after refactoring the UML model is also supported. Nevertheless, it is limited to work with UML (1.5) models and, though OCL evaluation is supported, validation of models is still too immature.

The EMF validation framework [66] provides a means to evaluate and ensure the well-formedness of EMF models both in batch and live modes. Although it is based on the definition of OCL constraints, the validation mechanisms have to be ad-hoc coded for each metamodel using the framework API. That is, you have to develop a plug-in that works over the plug-ins that implements your model editors. This was not a good option for us since we aimed at a more automatic and *declarative* way to add validation over M2DAT models.

In contrast, [101] showed a way to use the MDT-OCL project (another EMP subproject) for validation purposes doing exactly what we expected from a models validator. It used OCL syntax and it provided with models validation both in batch and *direct* mode. Nevertheless, this work showed two main shortcomings: when we used it to implement model validation in M2DAT prototypes:

- It is too dependent on EMF version, thus it did not work properly as soon as we updated EMF.
- The approach leans too much in EMF-generation capabilities what causes that, onde you have implemented some validation over your DSL, it results challenging to modify it. For instance, if some constraint has to be modified or added, the whole generation process had to be done again and the code generated modified by-hand.

In addition, we have to consider that, though it seems to be a rather simple language, OCL specification is quite complex. In fact, this is one of the reasons why you find such a variety of modelling tools having their own navigation language (ATL, QVT, MTL, XPand, Acceleo, JET, etc.) The most of them started from OCL since it was thought as a universal navigation and constraint language for MOF-based models. Actually, all of them ended by extending and adapting the standard to their needs. These shortcomings have a direct influence when OCL is used for validation purposes.

In the following we enumerate a number of issues that have been identified when OCL is used to that end [61, 102, 203]. They are mainly related with usability, but also with ease of development.

- OCL brings both expressiveness and limitations. You can write analysis expressions as complex as OCL allows to do it. In some cases it is more than enough while in others it is just poor. Just to put an example of this matter, think of the landscape provided by existing transformation languages. While they adopt an OCL style to define their navigation languages, none of them (apart from the standard QVT) uses OCL “as-is”. They extend the language to enhance its expressiveness.
- Directly related with the previous one, we might refer to the weak standard library of OCL. It hampers the specification of OCL constraint to carry out complex validations . For instance, there is a lack of useful operations to work with String types. Again, the fact that model transformation languages use to extend OCL to define their validation languages serves to prove this statement.
- Besides, OCL does not supplies mechanisms to provide with detailed reports regarding validation results. In the best case, you might alert of which is the violated invariant. Therefore, the user should know OCL to understand the error commited.
- By design, OCL does not provides with model modification capabilities. In particular, it cannot be used to create, update, or delete model elements, nor can it update attribute or reference values. As a consequence, there is no way of suggesting valid alternatives, what acts against usability, neither of executing corrective actions to solve the detected problem. For instance, if the name of an attribute is missing, we might show a widget to the user to set a name for such attribute.
- Besides, there is no distinction between severity levels. That is, your model might fulfil or not a constraint, but you can not state whether it is a minor problem (warning) or an issue that invalidates the whole model.
- OCL specification present inconsistencies, specially in the alignment with UML 2.X since there remains many references to UML 1.X. For instance, some classes specifications have been missing, such as TypeType or UnlimitedNaturalExp. This shortcomings hamper the development of OCL implementations.

- Since constraints to be checked use to be inter-dependent. It might occur that there is no sense in checking one constraint if another is previously does not evaluate to true. For instance, checking the uniqueness of the name of an user defined type has no sense if we have not checked before that every type owns a valid name.

As a conclusion, since OCL is closer to an implementation language than a conceptual language, one might argue about why not using another implementation language that extends OCL capabilities for validation purposes. Just as it has been done with navigation languages supported by current model transformation engines.

M2DAT follows such approach and uses EVL [201] (Epsilon Validation Languag,) to support validation of models for M2DAT supported DSLs. Among the different proposals studied, EVL is the only language that supports concepts such as dependent constraints, user interaction and the ability to define inconsistency-repairing behaviour (fixes). This way, we specify the constraints to be checked using EVL at metamodel level, and the Epsilon engine evaluates them (on demand) on every model conforming to such metamodel.

4.8 Development Process for M2DAT Modules

Once we have explained the main technical decisions that compose M2DAT technical design, this section aims at presenting the generics of using the specification in order to develop a new module for M2DAT, i.e. the technical support for integrating a new DSL in M2DAT.

The development process for new M2DAT modules is summarized in Figure 4-8. Steps in the development process are represented with rounded rectangles while the different software artefacts produced along the development process are represented with ellipses (or circles).

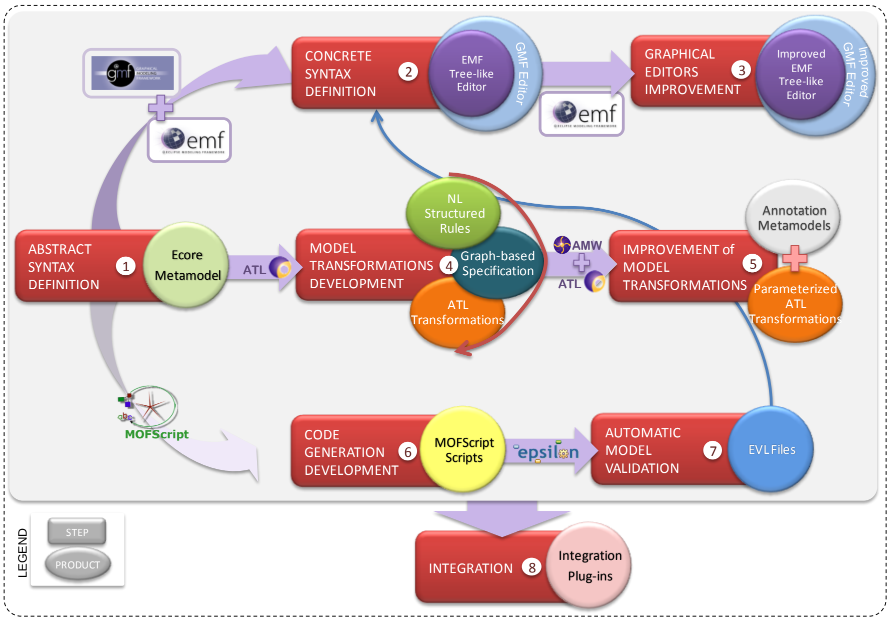

CONCRETE SYNTAX DEFINITION 2
EMF Tree-like Editor
GMF Editor
GRAPHICAL EDITORS IMPROVEMENT 3
Improved EMF Tree-like Editor
Improved GMF Editor
emf
ABSTRACT SYNTAX DEFINITION 1
Ecore Metamodel
ATL
MODEL TRANSFORMATIONS DEVELOPMENT 4
NL Structured Rules
Graph-based Specification
ATL Transformations
AMW
ATL
IMPROVEMENT of MODEL TRANSFORMATIONS 5
Annotation Metamodels
Parameterized ATL Transformations
MOFScript
CODE GENERATION DEVELOPMENT 6
MOFScript Scripts
epsilon
AUTOMATIC MODEL VALIDATION 7
EVL Files
INTEGRATION 8
Integration Plug-ins
LEGEND
STEP
PRODUCT

Figure 4-8. Development process for M2DAT modules

1. First step is the **abstract syntax definition of the DSL**. To that end, the metamodel of the DSL is defined in terms of the Ecore metamodel using the facilities provided by EMF (like the Ecore tools and the Ecore diagrammer).
2. Next task is the **concrete syntax definition of the DSL**. To that end, EMF and GMF capabilities are used to generate a couple of model editors, a tree-like editor with basic capabilities and a diagrammer. Due to GMF architecture, the latter is based on the former, what will help on subsequent steps of te process. In particular, the GMF editor uses the EMF.Core and EMF.Edit generated code (see section 2.1.12.2).
3. Since we have found that the EMF tree-like editor result quite convenient for accurate edition of models though it is too generic, next step implies the **graphical editors improvement** according to the techniques sketched in section 5.2.2.2. As a result, not only the EMF tree-like but also the GMF editors are improved.
4. Once the DSL has been defined, it is time to bridge it with already existing DSLs. To that end, the next consits of the **model transformations development**. Each transformation comprises several steps:
    - Defining a set of structured rules in natural language

- Translate then to a graph-based specification in order to formalize them with graph grammars
- Finally, translate the graph-based specification to executable trasformations with ATL. Actually, this is just a first draft of the transformation where all the possible design decisions are coded using default decisions.

5. Hence, the next step is the **improvement of the model transformations** introducing annotation models. The transformations are then modified to be able to process the weaving models containing the annotations. In addition, if the default annotation metamodel does not fit the requirements of the DSL in terms of annotations, **a new annotation (weaving) metamodel has to be defined**.
6. Besides, if the DSL is to be used to define models at the lower abstraction levels (PSMs), we have to address the **code generation**, i.e. to develop the model-to-text transformation/s that serialize terminal models into working-code using the MOFscript language.
7. Once all the transformations in which the new DSL have been developed, we address the implementation of the **automatic model validation**. We proceed this way because of some findings gathered during the development of first M2DAT's protoypes. We have found that as long as we were developing the model transformations, a number of domain rules that each input model has to obey arose. Indeed, these rules do not appear until the model transformations were addressed since the transformations execution failed in presence of erroneous models. Therefore, we delay the implementation of the support for automatic model validation until the last moment in order to be able to identify all the constraint that a given model has to satisfy. Note that the restrictions are coded at metamodel level and attached to EMF generated code, thus the validation could be invoked both from the EMF and GMF editors.
8. In addition, the DSL toolkit developed according to the development process described is basically a set of plug-ins. In order to ease the task of deploying such plug-ins into Eclipse we should address the **integration** of the developed module. To that end, another set of plug-ins have to be developed that let us publishing the new bundle of plug-ins (so-called features) that constitute the DSL toolkit.

Please note that Figure 4-8 is just a simplified version of the development process for new M2DAT's modules. For instance, we have not included the V&V tasks, neither the regression tests that should be made.

In the following we detail each one of the above-mentioned steps. To that end, we summarize the main features of each one of the selected components (in case they have not been presented previously) and show how they are used in the development process of any M2DAT module.

Moreover, when presenting the reference implementation for M2DAT in Chapter 5 we will show concrete examples of the application of the guidelines provided in the following sections.

4.8.1 Abstract Syntax Definition: using Ecore to define new Metamodels in M2DAT

The core of any project focused on MDSD is at the modelling languages it proposes. As well, the core of any modelling language is its abstract syntax. As section 4.2.4 stated, the definition of abstract syntaxes in M2DAT lies on EMF. Specifically, we use the basis of modelling technologies in Eclipse, the EMF's metamodelling language, so-called Ecore. We might say that Ecore is the MOF of Eclipse.

Note that this is an example of the everlasting dicothomy between standards and their implementations: EMF toolsmiths opted for defining their own metametamodel since MOF does not satisfy their needs. In fact, MOF evolution has caused its kernel, EMOF, to be aligned with Ecore, reverseing the natural tendency: the implementation is conditioning new versions of the standard.

This way, any new DSL to be incorporated in M2DAT will be defined in terms of the Ecore metamodel. Figure 4-9 shows a simplified subset of it. An *EClass* abstract the traditional concept of Class. EClasses own *EStructuralFeatures* that can be *EReferences* or *EAttributes*. The former represents the properties of the EClass while the latter represents association ends. The multiplicity of an association can be specified with the help of the *lowerBound* and *upperBound* attributes, while bidirectional relationships are expressed by two EReferences whith their *oppositeOf* reference pointing mutually to each other.

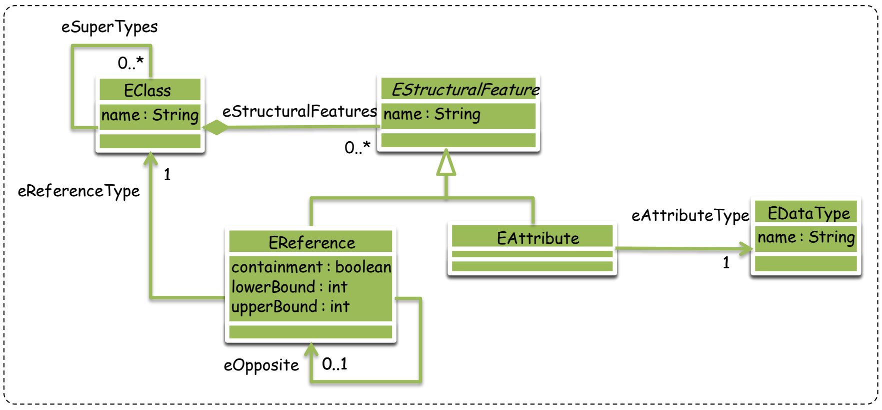


Figure 4-9. Simplified Ecore metamodel

Besides, EMF provides with the infrastructure to automatically connect the modelling concepts collected in the abstract syntax definition, with their implementation. From an Ecore model (i.e. an EMF metamodel), EMF generates the JAVA code to handle programatically instances of such model (terminal models). It also includes generic reusable classes for building editors for them. The EMF generation process is summarized in Figure 4-10.

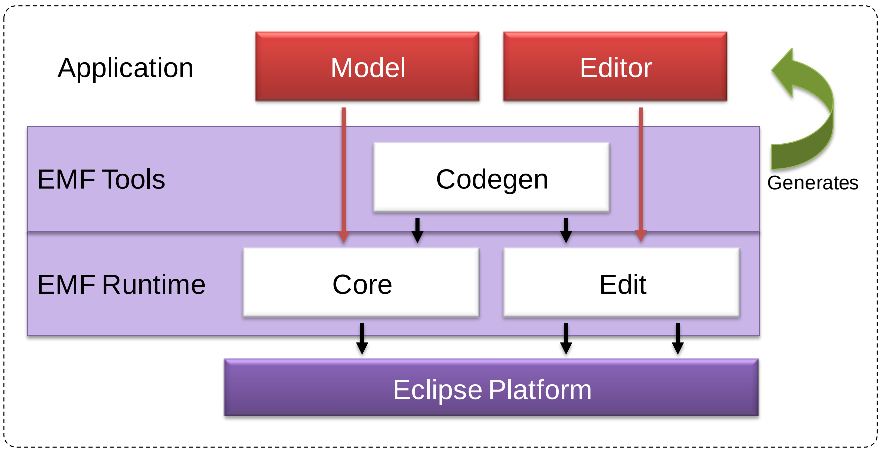


Figure 4-10. EMF Code Generation overview

EMF consists of three fundamental pieces: Core, EMF.Edit and EMF.Codegen.

- The Core provides the basic support for generating and executing the JAVA code that implement the model. Apart from the Ecore metamodel, it includes runtime support for the models, including change notification, persistence support with default XMI serialization, and an efficient API for manipulating EMF objects. The API lies in JAVA reflection to provide with a generic way of handling the objects.

- EMF.Edit includes generic reusable classes for building editors for EMF models and extends the Core by adding support for generating adapter classes that enable preview and work with the model, as well as a basic (visual) editor for the model.
- Finally, the EMF code generation facility (EMF.Codegen) is capable of generating everything needed to build a complete editor for an EMF model. It includes a GUI from which generation options can be specified, and generators can be invoked.

4.8.2 Concrete Syntax Definition: using EMF and GMF to develop Graphical Editors in M2DAT

This section summarizes the process for development of graphical editors thas has to be followed when building the technical support for a new DSL has to be integrated into M2DAT:

- First, the metamodel for the given DSL is defined using EMF. As mentioned before it will be specified in *Ecore* format (*MYDSL.Ecore* in Figure 4-11).
- From an Ecore (meta-)model, EMF provides runtime support for graphically editing, manipulating, reading, and serializing data based on the given (meta)-model. Thus, models conforming to the previous metamodel can be created using a very simple but powerful tree-like editor (e.g. see *Sample.mydsl* in Figure 4-11).
- As we have mentioned in section 4.3.3.3, we bet for improved EMF tree-like editors as the most convenient for handling models in M2DAT. Thus, after having generated the EMF-basic editor for our DSL, the next step is to follow the techniques described in section 5.2.2.2 to customize it according to the specific needs of our DSL.

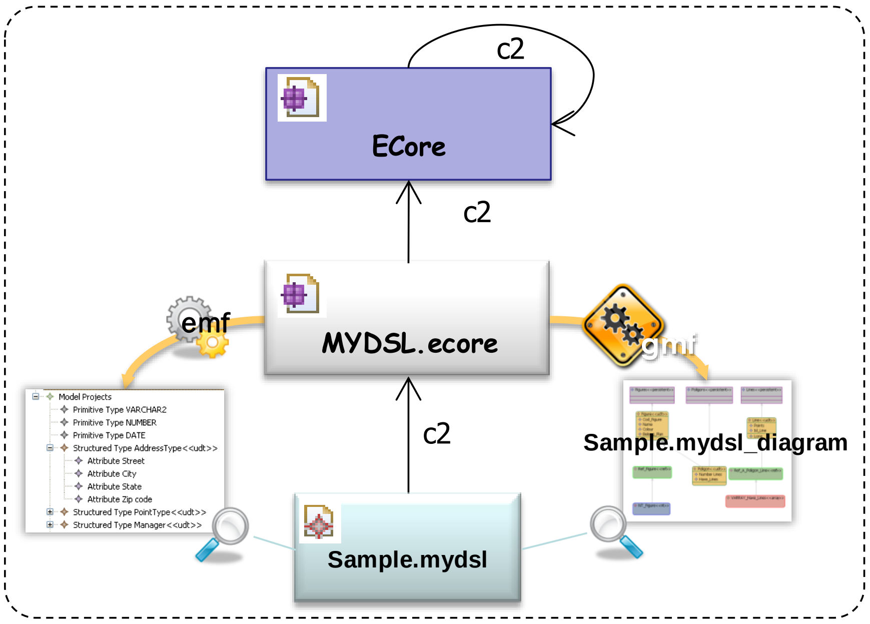


Figure 4-11. ECORE (meta-)models and EMF/GMF graphical editors

- Since we still think that diagrammers are convenient in order to provide with an useful overview of any model, GMF is used to develop a graphical editor for models conforming to the previous metamodel. Remember that bet for UML-like syntax, thus the GMF graphical model will be based on a common template defined to that end. This way, we ensure that all of the M2DAT editors share a common look and feel.
    - Please, note that the model is still stored in the same file (*Sample.mydsl*). In addition, a new file (*Sample.mydsl_diagram*) is created to store the graphical information about the classes and associations included in the model. Any subsequent change made over the graphical representation will be translated to the underlying model. Next Chapter will show some screen captures from the graphical editors already develop in M2DAT prototypes.

Finally, it is worth mentioning that these editors are automatically created as Eclipse plug-ins. Thus, they are integrated in the Eclipse platform without any extra effort.

4.8.3 Model Transformations Development in M2DAT

We have already introduced the way EMF facilities are used for defining a new DSL in M2DAT. To that end, we define the abstract syntax of every DSL using EMF metamodelling facilities and its concrete syntax using EMF and GMF capabilities for generating model editors.

The next step towards the integration of the new DSL is the development of the model transformations that will connect the models defined with such DSL with the rest of models already supported in M2DAT. To that end, we have to specify and implement the mapping rules that compose each model transformation.

Regarding how model transformations should be defined, the MDA guide [246] stated "the mapping description may be in natural language, an algorithm in an action language, or a model in a mapping language" (p. 24). This way, in [355] we sketched a common approach to address the development of model transformations in M2DAT:

- First, the mappings between models are defined using natural language.
- Next, they are structured by collecting them in a set of rules, expressed again in natural language.
- Then, the mapping rules from the last step are formalized using graph grammars [126].
- Finally, the resulting graph transformation rules are implemented using ATL.

4.8.3.1 Using graph grammars to formalize model transformations

This section focuses on explaining the use of graph grammars to formalize model transformations as a previous step to implementation oriented to give solution to some problems we have detected in the field of model transformations [74, 355].

There is a gap between the developers behind the different model transformations approaches (model transformation toolsmiths) and those who will have to use them, the researchers or developers working on MDSD methodological proposals (model transformation practitioners). The latter have to use the tools developd by the former to implement the mappings embedded in their proposals. The technique sketched in the previous section aims at reducing this gap by providing a simple methodological approach to the definition of mappings.

Moreover, formalizing the mappings before implementing them can be used to detect errors and inconsistencies in the early stages of development and can help to increase the quality of the models built as well as the subsequent code generated from them. These activities are especially important in MDE proposals since the models are to be used as a mechanism to carry out the whole software development process [249]. Graph grammars are based on a solid mathematical theory and therefore they present a number of attractive theoretical properties that allows formalizing model transformations.

In addition, from a pure mathematical point of view, we can think on UML-like models as graphs. A graph has nodes and arcs, while an UML model have classes and associations between those classes; this way the fact that models are well represented as graphs is particularly appealing to shorten the distance between modellers and model transformation developers Rule-based transformations with a visual notation may close the semantic gap between model transformation practitioners and toolsmiths [377].

To express model transformations by graph grammars, a set of graph rules must be defined. These rules follow the structure LHS:= RHS (Left Hand Side:= Right Hand Side). Both, the LHS and the RHS are graphs: the LHS is the graph to match while the RHS is the replacement graph. If a match is found on the source model, then it is replaced by the RHS in the target model. In the context of M2DAT, we will follow the approach introduced in [74] to define the graph transformation rules for each case.

In the following, we summarize its guidelines:

- The nodes in the LHS will be identified by consecutive numbers. These numbers make it possible to identify the respective nodes in the RHS.
- All the properties of the different nodes will have an initial value. To point out that this value is undefined, the term '???' is used.
- To refer to a LHS node in the RHS, the expression 'match(x)' will be used, being 'x' the number that identifies the node in the LHS.
- Likewise, when referring to an attribute of a LHS node, the dot notation will be used, for example 'match(x).name'.
- As the nodes in the LHS, the nodes in the RHS will be numbered. The next guidelines must be considered in relation with these numbers:
    - If the same number appears in the Left and the Right Hand Side, the type of the node in the RHS will be the same of the respective node in the LHS.

- If a node in the RHS is identified with a number followed by an apostrophe (x'), the type of this node will be different from the type of the respective node in the LHS. All the connections with other elements will be preserved.
- If a number appears in the LHS but not in the RHS, the respective node from the LHS will be deleted, as well as the connections in which it participated.
- If the number appears in the RHS but not in the LHS, a new node will be added.

4.8.3.2 Coding mapping rules with the ATL

Once the mapping rules have been formally specified using graph grammars, it is time to translate the formal specification into an operational abstraction. To that end, as we argued in section 4.4, we use ATL.

ATL is a model transformation language and toolkit that provides ways to produce a set of target models from a set of source models. Developed within the Eclipse platform, the ATL Integrated Environment (IDE) comprises a number of standard development facilities (syntax highlighting, debugger, editor, etc.) that eases the development of ATL transformations. It is mainly based on the OCL standard and it supports both the declarative and imperative approach, although the declarative one is the recommended.

Mappings are implemented in ATL by defining a set of rules: each rule specifies a source pattern and a target pattern, both of them at metamodel level. Once the ATL transformation is executed, the ATL engine establishes matchings between the source pattern and the source model. Then for each matching, the target pattern is instantiated in the target model, replacing the matching found in the source model.

In contrast with the most of exiting languages, ATL allows for rule inheritance and provides both implicit and explicit scheduling. The implicit scheduling is supported by the imperative constructions of ATL. When the transformation starts, the algorithm starts with calling a rule that is designated as an entry point and may call further rules. After completing this first phase, the transformation engine automatically checks for matches on the source patterns and executes the corresponding rules. Finally, it executes a designated exit point. Explicit scheduling is supported by the ability to call a rule from within the imperative block of another rule. ATL transformation descriptions are transformed to instructions for the ATL Virtual Machine, which executes the transformations. This is analogous to Java and the Java Virtual Machine.

One minor comment we would like to do about ATL is a problem detected when working with several models conforming to the same metamodel. In this situation, there is no way no distinguish between the elements of one model or other since the rules are defined at metamodel level. However, it would be helpful to be able to make such distinction in order to control which matchings will be found for each model. This can be done for instance in SmartQVT, where different identifiers can be used to refer to the same metamodel (we can do it also in ATL, but the engine will omit the distinction). Nevertheless, this drawback can be overcome in ATL by adding complexity on the guard of the rules.

4.8.4 Improvement of Model Transformations: Introducing Design Decisions in M2DAT transformations

This sections aims at summarizing the main issues related with the use of AMW to create annotation models to drive model transformation executions in M2DAT. To that end, it first introduces some insights on AMW to later present the technique to follow.

4.8.4.1 ATLAS Model Weaver

Since the definition of new weaving metamodels in AMW is based on the extension of the Core Weaving Metamodel [115], we first describe such metamodel. The Core Weaving metamodel, shown in Figure 4-12, contains a set of abstract classes to represent information about links between model elements.

- *WElement* is the base element from which all other elements inherit. It has a name and a description.
- *WModel* represents the root element that contains all model elements. It is composed by the weaving elements and the references to woven models.
- *WLink* expresses a link between model elements, i.e., it has a simple linking semantics. To be able to express different link types and semantics, this element is extended by different metamodel elements.
- *WLinkEnd* defines the link endpoint types. Every link endpoint represents a linked model element. It allows creating N-ary links.
- *WElementRef* elements are associated with a dereferencing function. This function takes as parameter the value of the *ref* attribute and it returns the linked element. For practical reasons, it is defined as a string attribute. There is also the inverse identification function that takes the linked element as parameter and that returns a unique identifier.

- *WModel*'s contains also *WModelRef's*, which is equivalent with the reference of *WLinkEnd* and *WElementRef*, but for models as a whole.

It is possible to associate the dereferencing/identification functions directly with the link endpoints. However, the use of separate *WElementRef* elements enables referencing the same model element by several link endpoints.

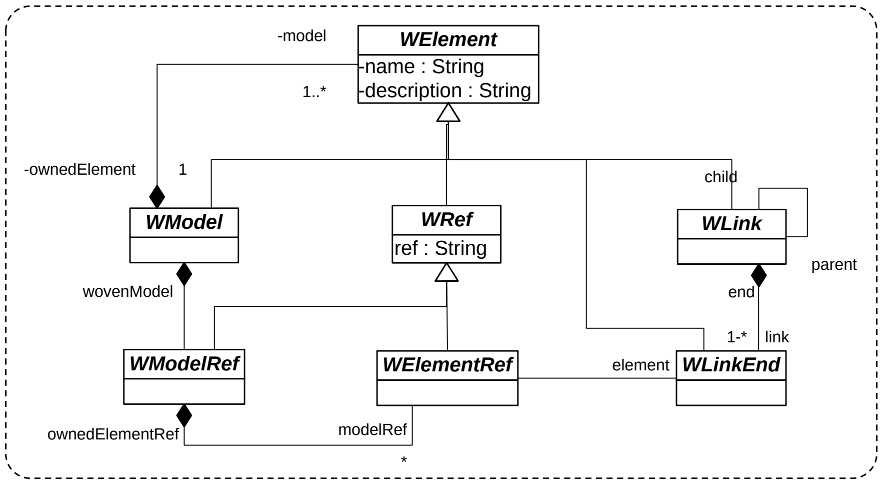


Figure 4-12. Core Weaving Metamodel

Typically, the classes from the core Weaving Metamodel are extended to define new weaving metamodels for specific contexts. One of those extensions was presented also in [115]. It is shown in Figure 4-13 below and allows defining annotation models. Note that the Core Weaving Metamodel is depicted on the top of the figure, whereas the extension is depicted on the bottom.

An annotation model includes a single-valued reference to the *AnnotatedModel* plus a set of annotation objects. Each annotation contains a single-valued reference to the model element plus a list of properties. The properties have an identification key and the corresponding value. The *AnnotatedModelElement* class acts as the proxy for the linked/annotated elements. That is, each record is merely a set of key-value pairs.

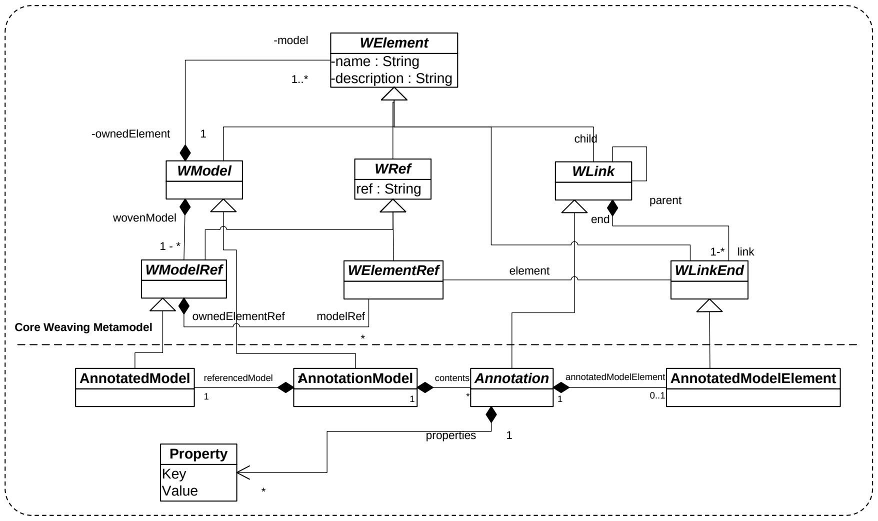


Figure 4-13. AMW Annotation Metamodel

Once you are able to define your own annotation metamodels, next step is showing how a weaving model is used in the model ttransformation development.

4.8.4.2 Using weaving models as annotation models on M2DAT

To address the development of model transformations in M2DAT we follow the method sketched in section 4.4.2. That is, we firstly carry out a preliminary study to obtain a set mapping rules expressed with natural language to later formalize them using graph grammars. The next step is to implement those formalized rules. To that end we use ATL.

However, as we have stated at the beginning of this section, in occasions some design decisions has to be considered before executing a model transformation. This was the case of some of the mappings embedded in M2DAT. To solve this drawback we use AMW weaving models as annotation models.

All this given, the resulting process to code model transformations in M2DAT is summarized in Figure 4-14. For every execution of the ATL transformation - in other words, for each source model (Ma) - we define a weaving model (Annotation Model) conforming to the annotation metamodel that in turn conforms to the Core Weaving Metamodel. Such weaving model contains a set of annotations. They represent the extra information needed to execute the transformation (we may refer to them as the *parameters* of the transformation). Thus, the target model (Mb) is generated from the source model and the weaving model. This process allows obtaining different target models from the very same source model just by modifying the annotation/weaving model.

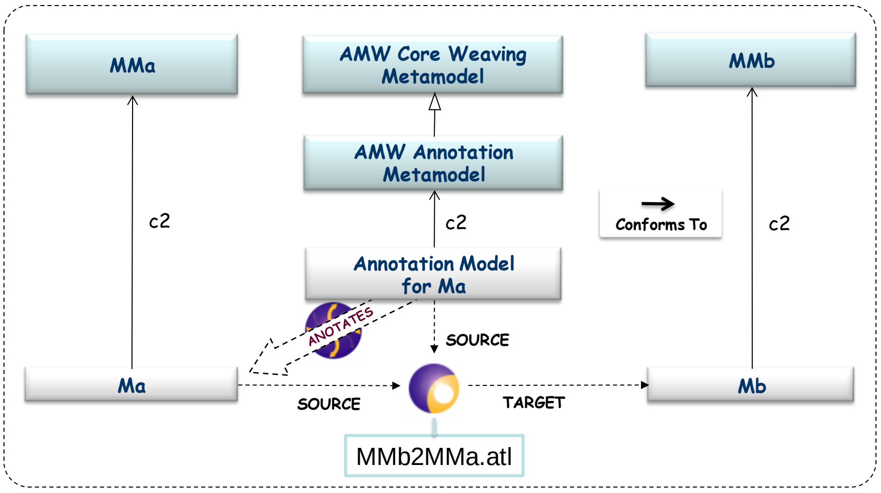


Figure 4-14. Using Weaving models as annotation models to drive model transformations execution

It is worth mentioning that, though the annotation metamodel presented works pretty fine for many scenarios, in some cases we have to define an ad-hoc annotation metamodel to ease the addition of extra information to drive the mapping process. For instance, we proceed this way to develop the model transformations encoded in [357].

We may qualify this technique of simple due to the genericity and power of the AMW tool, its good coupling with the ATL model transformation solution and the use of a common underlying framework used as model handler by all the technical solutions that compose M2DAT: EMF.

4.8.5 Code Generation: model-to-text transformations in M2DAT

There is not much to say about how code generation tasks are to be tackled in M2DAT, therefore in this section we will limit ourselves to introduce the main features of the MOFScript language, the model-to-text transformation language we have used to implement code generation in existing M2DAT's modules.

MOFScript is a prototype implementation based on concepts submitted to the OMG MOF Model to Text RFP process [267]. Since it was the first submission to the OMG RFP, it is probably the most contrasted and the most commonly used. Besides, its training period is quite short. After coding some model-to-model transformations, moving to model-to-text transformations is quite easy. A more detailed explanation on the way MOFScript is used will be given when presenting the case study. Opposite to the declarative approach of ATL (and the vast majority of existing model to model proposals), model to text transformation engines take the form of imperative programming languages. In fact, a MOFScript script is a parser for models conforming to a given metamodel. While it parses the model structure, it generates a text model based on transformation rules. On a second phase this text model is serialized into the desired code. This way, MOFScript takes advantage of the metamodel to drive the navigation through the structure of the source model, just as an XML Schema drives the validation of an XML file. As a matter of fact, every model is persisted in XMI format, an XML syntax for representing UML-like (or MOF) models. All things considered, model to text transformation are much simpler than model to model transformations.

4.8.6 Automatic Model Validation: supporting Model-Checking in M2DAT with EVL

Finally, this sectiom aims at summarizing how the specification of DSLs in M2DAT is to be completed by means of defining the set of additional constraints that every model should satisfy in order to be considered a valid model. According to section , automatic model validation is implemented in M2DAT using EVL, one of the languages provided by the EPSILON componente.

EPSILON [203] (Extensible Platform for Specification of Integrated Languages for mOdel Management) is an Eclipse component that provides support for a number of tasks related with model-driven development. To that end, it integrates a family of languages for specialized tasks, like models merge or model comparison.

The Epsilon Validation Language [201] (EVL) is one of them. In particular, EVL is a language to specify and evaluate constraints on models of arbitrary metamodels and modelling technologies. The idea is the usual, you specify the constraints to be checked at metamodel level in an EVL file or module. Later, these constraints are evaluated (on demand) over conforming models.

EVL uses an OCL-like syntax. Indeed, EVL validation specifications are structured into Invariants and each Invariant is applicable only over the objects whose type conforms to the one specified in the Context of the invariant. This way we can look at EVL as OCL with annotations that provide with additional facilities:

- **Guards** to restrict the context of a given invariant (that is, not all the association objects, but just those whose name is X)

- **Fixes** that allows user interaction. A fix let you specify a message to show the user when the invariant is not fulfilled, as well as valid alternatives to repair the problem. The latter is implemented using EOL (Epsilon Object Language, [201]), another OCL-based language to navigate and modify models. In fact, EOL is the core language of EVL. Notice that this way, the validation mechanism implemented in the DSL toolkit already incorporates the facilities for updating the model to solve the issue that raised the error.
- Two different sub-types of Invariant (**Constraint** and **Critique**) to allow the separation between errors, that invalidate the model, and warnings, that are allowed but act against the quality of the model.

To show how EVL works, Figure 4-15 shows a simple example: a Critique to prevent from Classes whose name does not start with an upper case.

```
context Class {

  -- The name of a class should start with an upper case letter
  critique NameShouldStartWithUpperCase {

    guard : self.satisfies('HasName')

    check : self.name.substring(0,1) = self.name.substring(0,1).toUpperCase()

    message : 'The name of class ' + self.name + ' should start with an upper-case letter'

    fix {
      title : 'Rename class ' + self.name + ' to ' + self.name.firstToUpperCase()

      do {
        self.name := self.name.firstToUpperCase();
      }
    }
  }
```

Figure 4-15. Simple EVL example

The context of the Invariant is Class, thus it will be evaluated over every class found on the model. Note that this is not correct indeed since there is also a guard. The guard limits the set of objects over which the body of the invariant will be evaluated. In this case, just those Classes for which the 'HasName' invariant evaluates to true (i.e. those that have a name). The Check defines the body of the invariant, i.e. whether the class name starts with an upper case. The Message specifies the information provided if the check evaluates to false, while the Fix defines a context-aware title ('Rename class …') and contains a statement block to specify the fixing functionality (Do part).

Validation: M2DAT-DB

A reference implementation is to be used as a consistent interpretation for the corresponding specification. Indeed, at least one relatively trusted implementation of a given specification is nedeed to discover errors or ambiguities in the specification, and validate the feasibility of the underlying proposal [95]. Thereby, the main features of a reference implementation are [100]:

- Developed concurrently with the specification.
- Verifies that specification is implementable.
- Serves as a reference against which other implementations can be measured.
- Helps to clarify the intent of the specification.

This Chapter aims at introducing the reference implementation for M2DAT: M2DAT-DB, a set of interconnected modules developed according to M2DAT's specification. All together they conform the technical support for MIDAS/DB [363], the MIDAS proposal for the development of the content aspect of a WIS. In particular we will focus on the module that supports a DSL for modelling ORDB schemas conforming to the SQL:2003 standard and the model transformations in which it is implied.

The construction of M2DAT-DB serves as reference implementation for M2DAT specification since it confirms that the specification is implementable and clarifies the way it has to be done.

To that purpose we start by giving a brief overview on M2DAT-DB architecture and capabilities to later focus on how each MDE task in the development of the afore-mentioned DSL is addressed.

5.1 M2DAT-DB Overview

M2DAT-DB is a framework for model-driven development of modern DB schemas that support the whole development cycle, from PIM to working code. In particular, M2DAT-DB support the generation of ORDB schemas for Oracle and the SQL:2003 standard as well as XML Schemas from a conceptual data model represented with a UML class diagram.

However, we do not want to focus on M2DAT-DB itself as a development tool but as the first prototype of M2DAT. That is, apart from providing with the mentioned functionality, M2DAT-DB has served to prove that M2DAT architecture and design decisions were right and to put them into practice. In the following we introduce M2DAT-DB architecture as well as the functionality supported by the tool.

5.1.1 *M2DAT-DB architecture and capabilities*

Figure 5-1 provides an overview of M2DAT-DB and the model-driven development process for modern DB schemas that it supports.

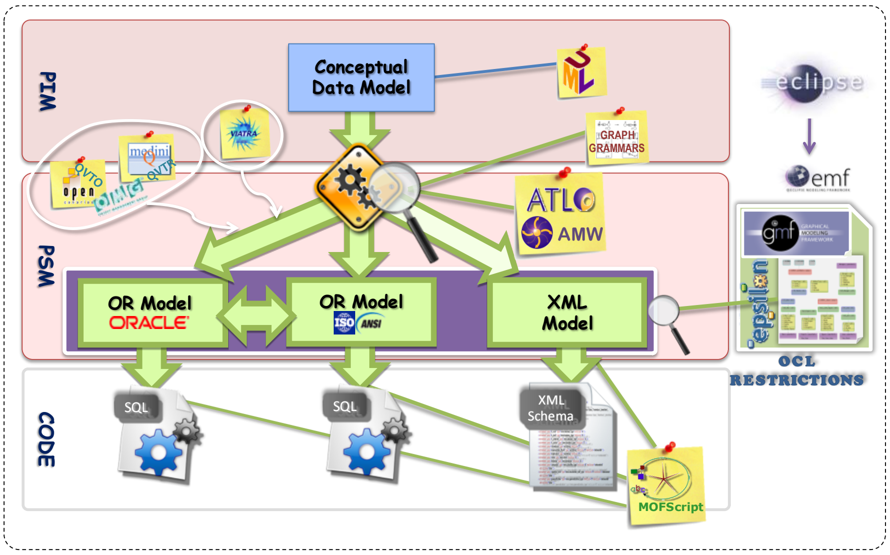


Figure 5-1. M2DAT-DB Architecture

A conceptual data model serves to model the DB schema at PIM level. It is represented by means of a UML class diagram. This model is defined using the tools provided by UML2 and UML2 Tools, two subprojects of the Eclipse Modelling Tools project (MDT, http://www.eclipse.org/modeling/mdt/). They focuses on providing implementations of industry standard metamodels, as well as exemplary tools for developing models based on those metamodels. This way, the UML2 project provides with an EMF-based implementation of the UML standard [391], while UML2 Tools is a set of GMF-based editors for viewing and editing the different types of UML diagrams.

At PSM level, two different technologies are considered to implement the DB schema: Object-Relational and XML. This way, the DB schema will be modelled with an ORDB model or an XML Schema model. In turn, two different OR models are considered, the one for the standard, SQL:2003 [387] and the one for an specific product, Oracle 10*g* [391].

To move from the PIM to the desired PSM, three model-to-model transformations have been developed following the method for model transformation sketched on Section 4.8.3. In particular, we have developed the following transformations:

- From UML class diagram to ORDB model for Oracle (UML2ORDB4ORA)
- From UML class diagram to ORDB model for SQL:2003 standard (UML2SQL2003)
- From UML class diagram to XML Schema model (UML2XMLSchema).

All of them were first defined in set of structured rules, next formalized by means of graph grammars and finally translated to ATL mapping rules.

Note that, in order to evaluate different languages for model transformation, the UML2ORDB4ORA transformation was replicated using QVTo (the QVT-Operational Mappings implementation from OpenCanarias, see section 2.3.3.13); VIATRA (see section 2.3.3.10) and mediniQVT (see section 2.3.3.12). Some highlights gathered during the development of such transformations were presented in section 5.3.4.2.

The mapping from conceptual data models to DB schema models leaves some space to design decisions. For instance, which collection type is to be used when mapping multivalued attributes. To support the introduction of those decisions, we use weaving models as annotation models, according to the process described in section 4.8.4.2. This way, the ATL existing transformations were refined to compute not only the source models, but also such AMW annotation models.

At PSM level, we have also built the bridge to move from the SQL:2003 ORDB model to the one for Oracle and vice versa. To that end, we have developed two ATL unidirectional transformations (SQL20032ORDB4ORA and ORDB4ORA2SQL2003) since support for bidirectional transformations is still quite immature [99].

Finally, a last set of MOFScript model-to-text transformations generates the working-code from each specific PSM, i.e. SQL *standard* from the SQL:2003 model, SQL for Oracle from the Oracle model and XML Schema from the XML Schema model.

As well, three diagrammers plus three tree-like editors have been developed using EMF facilities. One for each type of PSM supported. Actually, as we discussed in section 4.3.3.3, we prefer the tree-like editors for development tasks, though the diagrammers are well-suited to provide with a quick overview of the model. In this sense, it is worth mentioning that we discarded the diagrammer for XML schema models since we realized that, as long as the model get complex, its representation as a class diagram was unmanageable. Indeed, the tree-like EMF editors fit better to the nature of XML documents and result much more intuitive and user-friendly for this task.

Finally, as we have already mentioned, M2DAT uses EVL to support automatic model validation (see section 4.7.2). This way, EVL files were coded for each DSL integrated into M2DAT-DB. These files collect the set of restrictions that have to be checked over a terminal model defined which any one of such DSLs.

As Figure 5-1 shows, we plan to add support for two other MDE tasks: textual editing of models and extracting models from legacy code. In fact, we have already developed a textual editor for Oracle OR models. To that end we have used the TEF framework (Textual Editing Framework, see section 2.2.14). Though the results are promising, the framework is still too instable to be included as is in M2DAT-DB. Regarding model extraction, we have started to study text-to-model transformation languages (see Appendix C) to evaluate if they fulfil our requirements in order to integrate model extraction capabilities into M2DAT.

All things considered, it becomes clear that developing M2DAT-DB implies making use of the whole M2DAT's solution defined in Chapter 4. However, next section provides with some ideas to back up the election of M2DAT-DB as a reference implementation for M2DAT.

5.1.2 Why we choose M2DAT-DB as a first M2DAT prototype

When we addressed the task of defining and building a MDSD framework to support the development of WIS, the first task was designing its architecture. To that end, we fixed some requirements, like modularization and extensibility.

Once we had a first draft of the architecture, it was time to validate it. To that purpose, we built the two MIDAS-CASE prototypes presented on Chapter 3. Such prototypes serve to confirm that, though they performed their jobs in an efficient way, they presented some drawbacks from a pure MDE point of view. They implemented a very localized functionality. Each prototype supported a different DSL, whose metamodel was not too complex, in a completely isolated way. In fact, when we started thinking on the connection of MIDAS-CASE prototypes, we realized that MIDAS-CASE architecture did not meet our needs. As well, they provided with a set of lessons learned.

As a result, we defined a new version of the architecture, the M2DAT architecture presented in section 4.1. Regarding the conceptual architecture, it was merely a refinement of the previous MIDAS-CASE architecture. In contrast, technology advances resulted in a completely new technical design as we have shown. Indeed, all the tools and components used in M2DAT's specification did not exist when we developed MIDAS-CASE.

To validate M2DAT's specification, we had to make a decision on which of the methodologies that integrate MIDAS (see Section 1.3) was the most suitable to develop a first prototype supporting the method. In contrast with MIDAS-CASE, this time we aimed at testing every one of the capabilities that we wanted to integrate into M2DAT. Therefore, we chose to develop the technical support for MIDAS/DB, the method for the content aspect of MIDAS [363] since:

- First of all, it was complex enough since it comprises a number of different DSLs.
- It allowed us to prove the feasibility of the proposal for a complete development cycle: from PIM models to working-code.
- Different platforms were to be targeted, two standard platforms, SQL:2003 and XML Schema, plus a commercial one, Oracle. In addition, both the result of the code generation processes for XML Schema and Oracle could be loaded and validated against existing commercial products.
- Likewise, the models involved were real models, widely-acknowledged and rather complex. XML Schema and SQL:2003 are two standards widely used, whereas Oracle is probably the most adopted DBMS worldwide.
- It included not only PIM2PSM, but also PSM to PSM transformations plus model-to-text transformations. Besides, the complexity of those models imply the need to support additional mechanisms of validation to enforce the consistency of terminal models.
- Last but not least, before addressing the development of this thesis, the research activities of the PhC candidate were focused on the study of OR and XML databases. In particular, we had been working in the definition of a methodological proposal to model ORDB schemas and XML schemas. Therefore, we were ready to tackle the definition of DSLs for these tasks, plus the development of the corresponding toolset to support them.

All in all, the rest of this Chapter presents how the specification of M2DAT has been put into practice to built M2DAT-DB, the technical support for the development of the content aspect of a WIS. To that end, we use the support for each MDE tasks that comprise a MDSD process. As we have mentioned, we will focus on the development of a DSL for modelling ORDB schemas conforming to the SQL:2003 standard.

As well, along this Chapter we will show the application of the resulting tooling. To that end, though we have handled a battery of self-made case studies during the development process, here we will not use one of them. Instead, we take one from existing literature to illustrate that the tooling developed works properly for any model. Using an "external" case study prevent us from using ad-hoc models that might fit better to our needs. So, to illustrate the following sections we use a case study taken from [385] (p. 5): an Online Movie Database (OMDB). The complete Case Study can be found in Appendix D.

5.2 Defining new DSLs in M2DAT

This section focuses on introducing how the definition and toolset building for a new DSL is addressed in the context of M2DAT. In essence, this task corresponds to the definition of a new metamodel, that collects the abstract syntax of the DSL, and the construction of an editor for the DSL, that associates the concepts collected in the metamodel with its concrete syntax.

In the following we present the definition of the SQL:2003 DSL that allows modelling ORDB schemas conforming to the SQL:2003 standard.

5.2.1 Abstract Syntax Definition

M2DAT's metamodels are defined in terms of Ecore (see section 4.2.4), the metametalanguage of EMF, a simplified implementation of EMOF (Essential MOF, [265]). We have already presented the Ecore metamodel in section 2.1.12.2

In the following we present how the Ecore metametamodel is used to define the OR metamodel for SQL:2003 standard.

5.2.1.1 ORDB Metamodel for SQL:2003

Figure 5-2 shows the complete ORDB metamodel for SQL:2003. Due to its complexity, in the following we have shred it according to the main building blocks that contains to ease its presentation.

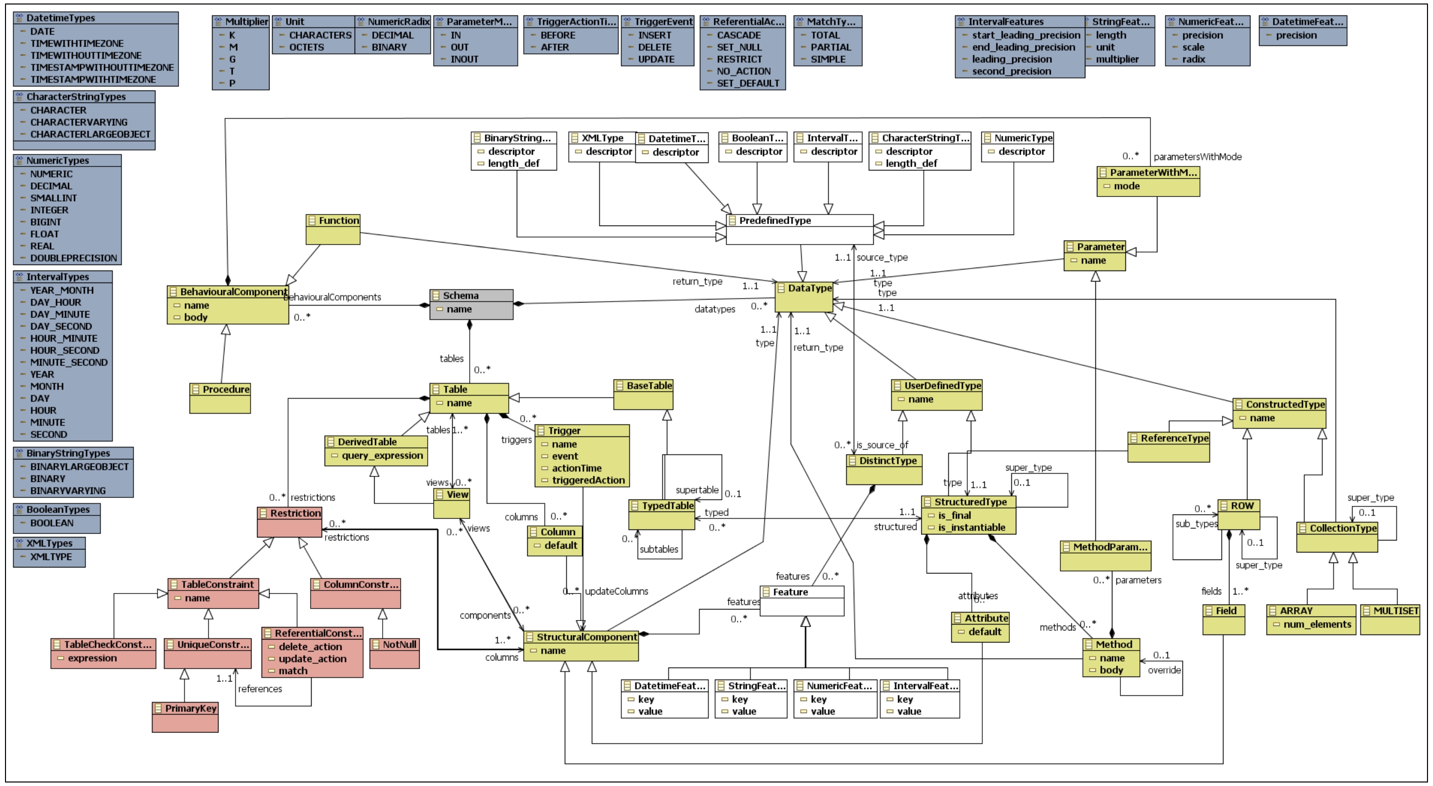


Figure 5-2. SQL:2003 ORDB Metamodel

First of all, note that due to the underlying XML nature of Ecore, any Ecore metamodel has to include a root element. In this case, the root element is the **Schema** class.

Each Schema is composed of (Figure 5-3): **DataTypes**, whether they are built-in (predefined) or user-defined types; **Behavioural Components**, that will be **Procedures** or **Functions** (returning an object of a DataType) and ***Tables***.

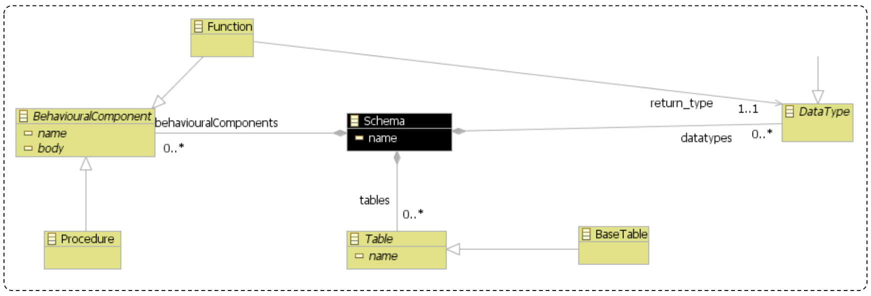


Figure 5-3. Partial view of the ORDB metamodel for SQL:2003: Schema metaclass

Regarding Data Types, we have identified three big groups (see Figure 5-4):

- **Predefined Types** receive special attention, thus section 5.2.1.2 is dedicated to the technique devised to support modelling of built-in types in PSM models.
- ***User Defined* types** could be **Distinct Types**, defined over a Predefined Type, or **Structured Types**, the basis of ORDB schemas designing.
- Finally, constructors allow defining Constructed Types on top of Data Types. This, way a **Reference Type** simulates a pointer to a User Defined type. A ***Collection*** serve to model sets of objects of a particular Data Type. They could be **ARRAY**s (predefined size) or **MULTISET**s (whose size can be modified dynamically), and **ROW** types that collect a set of fields, all of them of a Predefined Type.

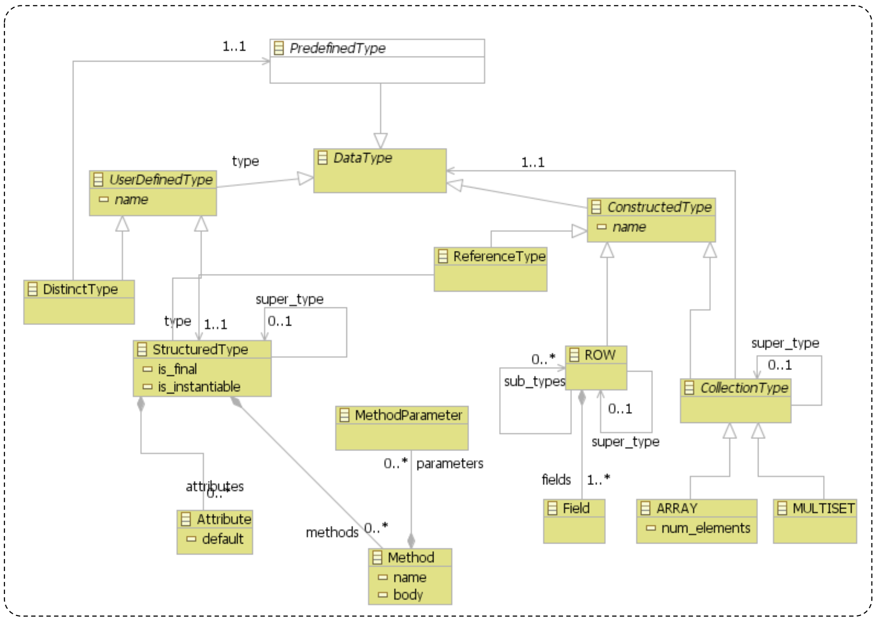


Figure 5-4. Partial view of the ORDB metamodel for SQL:2003: Data types

Every Structured Type (Figure 5-5) may extend another Structured Type. Besides, it owns a set of **Attributes**, that admits a default value and a set of **Methods**. Each method could override another one and contains a set of **Parameters**, that will be a **Parameter With Mode**, i.e. in, out in/out mode.

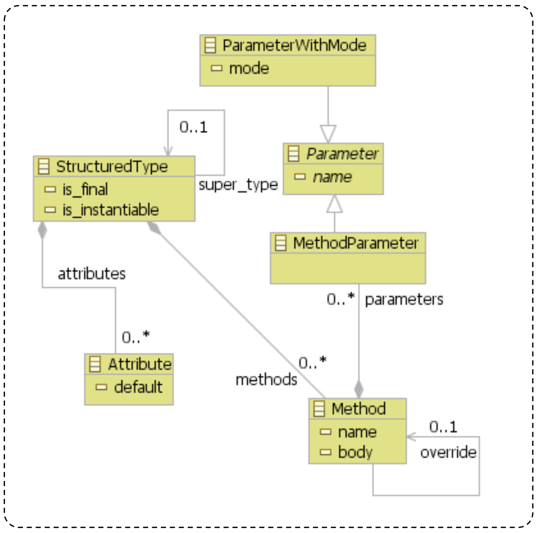


Figure 5-5. Partial view of the ORDB metamodel for SQL:2003: Structured Type

The **Structural Component** class (see Figure 5-6) collects the set of properties and relationships shared by **Columns** (belonging to a particular Table), **Attributes** (belonging to a particular Structured Type) and **Fields** (belonging to a particular Row Type). The particularities of every Structural Component are modelled as **Features**. For instance, the model could contain a Column object of CHAR type that includes a Feature object that limits the size of the column to 20 characters.

Though every **Restriction** is always included in a Table, it is related to one or more Structural Components. In turn, there are **Table Restrictions** and **Column Restrictions**. The latter will be **Not Null** constraints, while the former could be a **Check**, a **Referential Constraint** (Foreign Key) or a **Unique** constraint, that could be as well a **Primary Key**.

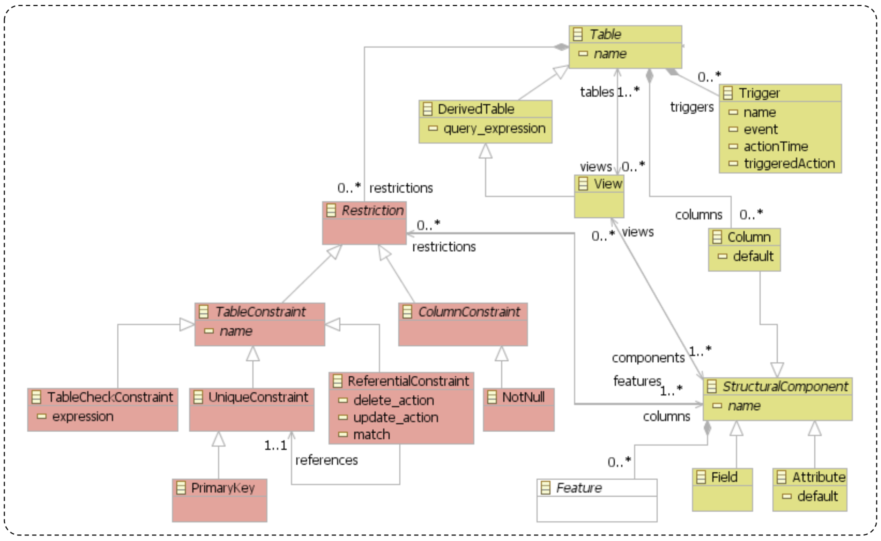


Figure 5-6. Partial view of the ORDB metamodel for SQL:2003: Structural Component and Restrictions

To conclude, we will focus on the different types of Tables that could be found on a SQL:2003 ORDB schema.

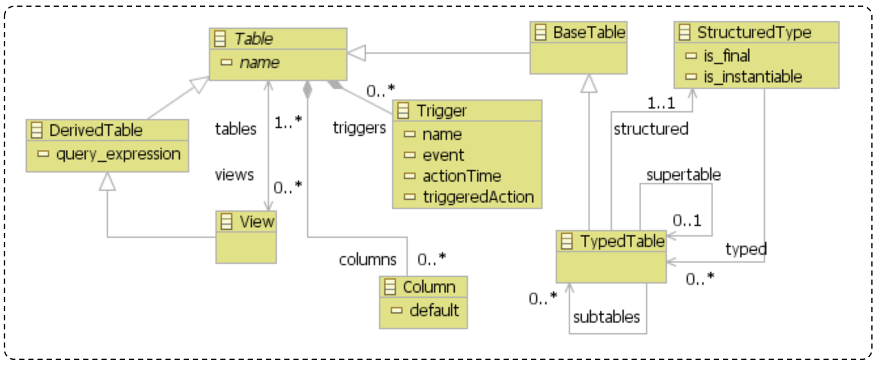


Figure 5-7. Partial view of the ORDB metamodel for SQL:2003: Tables

A **Base Table** is a regular table whereas a **Typed Table** is a special type of Base Table defined over a Structured Type. While a Base Table contains values, a Typed Table contains objects. Besides, it could extend another Typed Table. On the other hand, a **Derived Table** is defined over a Table and may be persisted as a **View**. Derived Tables are created on-the-fly using a SELECT statement, and referenced just like a regular table or view. Derived tables exist in memory and can only be referenced by the outer SELECT in which they are created. For instance, *SELECT * FROM (SELECT * FROM Sales) AS a* . Finally, triggers might be defined over any table.

5.2.1.2 Modelling Primitive Types on Platform Specific Models

As we have sketched in the previous section, the task of modelling built-in types in PSMs needs special attention. If you aim at being able of generating working code from a model, you need it to be very detailed. Otherwise, you end up generating just some skeleton of the final code. Part of the complexity related with platform modelling resides in the type system supported by each platform. Technological platforms, like SQL:2003 or Oracle, supports very rich type systems. To be able to use the whole type system supported by a platform when defining a model for such platform, special considerations have to be made when defining the metamodel of the corresponding DSL. In addition, related tooling has to provide with special facilities to simplify the definition of models. The tooling issue will be addressed later. Here we focus just on the technique devised to include built-in types in the metamodel in an efficient and semantically rich way.

Each platform uses to structure the supported built-in types in a hierarchical way. Leafs are the concrete types that can be instantiated, while enumerated data types serve to choose one among the family of final types. For instance, if you have a look at Figure 5-8 you will find that there are three different Number Types, NUMBER, BINARY FLOAT and BINARY_DOUBLE.

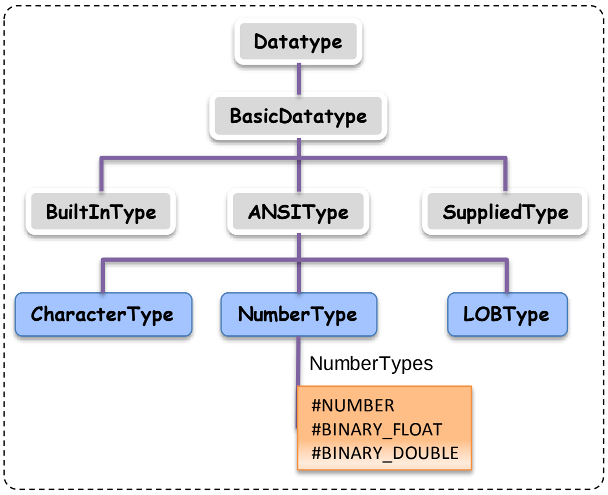


Figure 5-8. Partial view of the SQL:2003 Built-in Data Type System

Each different Data Type owns a series of inherent characteristics, that no other Data Type has. When the Data Type is used to define the type of some element in a model, a value has to be set for each characteristic of the Data Type. This value applies just for this very concrete use of the Data Type. If the Data Type is used to define the type of another element, the value of each characteristic has to also set for that concrete use. For instance, in Figure 5-9, the Customer table owns two columns, Name and Address, having the same type, CHARACTER. However, the size of the CHARACTER Data Type has a different value for each column.

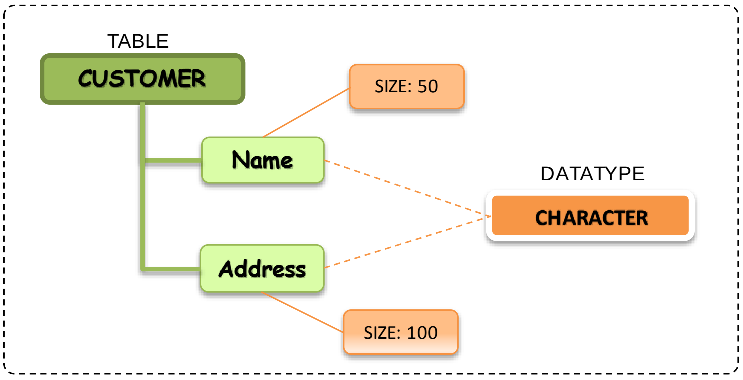


Figure 5-9. Defining Data Types characteristics

To support the complete modelling of built-in Data Types without adding too much complexity to the metamodel we lean on two main techniques:

- A **Feature class** is added in the metamodel. It is extended to define a set of valid Features for each family of Data Types. To that end, each descendant is a pair key-value, where the key take its value from an enumerated Data Type that states which features can be defined for each concrete Data Type. For instance, Figure 5-10 shows the Features defined for the SQL:2003 built-in Data Types. This way, a Structural Component whose type is Numeric, may include a Feature object that sets the precision, the scale and the radix of the concrete Data Type used.

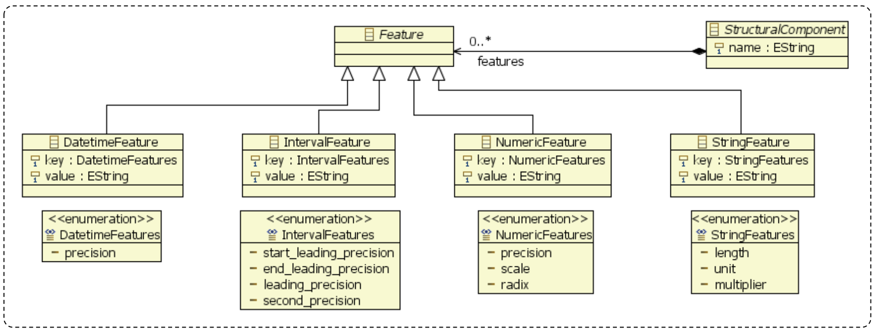


Figure 5-10. Partial view of the ORDB metamodel for SQL 2003: Features

- Then, **each Structural Component owns a set of Features**. This way, when a Structural Component object is added, the value of the features for the Data Type used to define the type of the object are nested in theoobject itself. An example is shown in Figure 5-11: both, the Name and Address attributes of the Customer table share the same data type: Character. However, each one “customize” the data type according to its needs. In this case, the size of each attribute needs to be different. To that end, each one owns a feature object. The key value is taken from an enumerated data type defined to that purpose. The value of the feature is the size of each attribute. Section 5.2.2.1 will show how this metaclasses are used/instantiated in M2DAT’s editors.

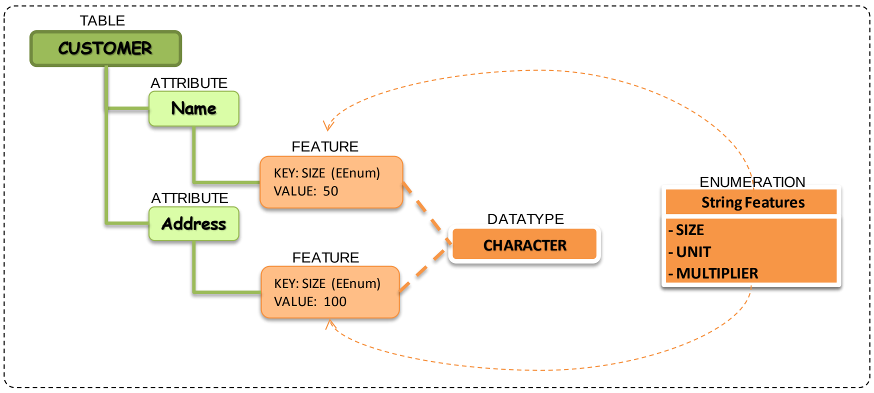


Figure 5-11. Using features to model built-in data types

- Besides, including metaclasses to model the whole set of built-in data types blots out the metamodel. To avoid this problem, we elliminate all the leaf types by using *Descriptors* to distinguish between the concrete types that compose a particular family of primitive Data Types. This way, each family is modelled by adding just one metaclass. Such class contains a Descriptor attribute whose value is defined by an enumerated Data Type whose values are correspond to the types that compose the family of Data Types. For instance, Figure 5-13 is a partial view from the SQL:2003 ORDB metamodel that shows the built-in data types. Following the technique described, the *CharacterStringTypes* enumerated data type indicates that there are three different Character String types: CHARACTER, CHARACTER VARYING and CHARACTER LARGE OBJECT.

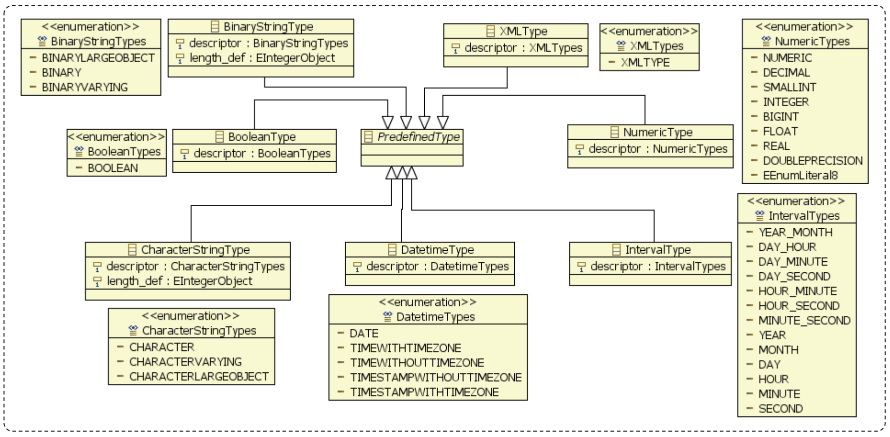


Figure 5-12. Partial view of the ORDB metamodel for SQL 2003: Built-in Data Types

5.2.2 *Concrete Syntax Definition*

From the .Ecore file that collects and EMF-based metamodel, i.e. the abstract syntax, EMF allows generating a tree-like editor with basic capabilities for models conforming to the metamodel. Besides, GMF allows generating a graphical editor based on boxes and edges (diagrammer) from that same metamodel. To that end, you have to define three additional models that encode the relationships between metaconcepts and graphical elements (see sections 2.1.12.2 and 4.8.1).

As we have already mentioned, from our experiences working with both type of editors, we conclude that the tree-like editor is best suited for development tasks, whereas the graphical one provides with a comfortable overview of the depicted model.

Thus, although we have developed the graphical editors for M2DAT models, we have focused on identifying the way to boost and customize the tree-like editors of EMF. We present the results using the editor for the SQL:2003 ORDB DSL in the following sections. First we give a brief introduction on EMF support for automatic editors generation.

5.2.2.1 EMF Implementation

Section 2.1.12.2 already gave an overview on the use EMF for metamodelling purposes. Here we will give a brief overview on the insights of tree-like editors generation in EMF.

Once we have defined our metamodel, the first step is the creation of an EMF model (so-called Genmodel) from our Ecore model, also known as the core model. This is a mandatory step previous to code generation for our model.

Most of the data needed by the EMF generator is stored in the core model. The classes to be generated and their names, attributes, and references are all there. There is, however, more information that needs to be provided to the generator, such as where to put the generated code and what prefix to use for the generated factory and package class names, that isn't stored in the core model. All this user-settable data also needs to be saved somewhere so that it will be available if we regenerate the model in the future. The EMF code generator uses a generator model, the Genmodel, to store this information.

The significance of all this is that the EMF generator runs off of a generator model instead of a core model; it's actually a generator model editor. When you use the generator, you will be editing a generator model, which in turn indirectly accesses the core model from which you're generating. Thus, the .genmodel file is a serialized generator model with cross-document references to the .Ecore file. In summary, as showed in Figure 5-13, the Genmodel is an EMF model that wraps the core model. Generator model classes are Decorators of Ecore classes.

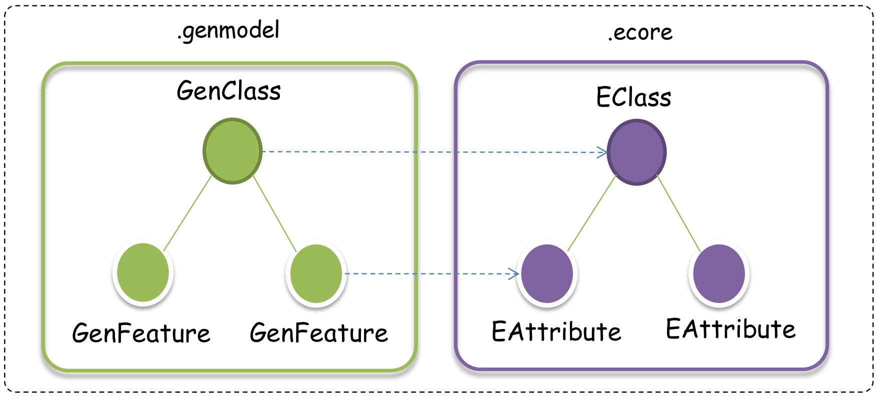


Figure 5-13. Relationship between .genmodel and .Ecore model

Separating the generator model from the core model like this has the advantage that the actual Ecore metamodel can remain pure and independent of any information that is only relevant for code generation. The disadvantage of not storing all the information right in the core model is that a generator model may get out of sync if the referenced core model changes. To handle this, the generator model classes include methods to reconcile a generator model with changes to its corresponding core model. Using these methods, the two files are kept synchronized automatically by the framework and generator.

From the Genmodel, EMF generates JAVA code that can be structured in three big categories: the model code, the edit code and the editor code. As well, test code could be generated but it is rarely used for anything. See

In essence, the Model code allows accessing the metamodel, create a conforming model and serialize and de-serialize it programmatically. This code is used by the Edit and Editor code, that wraps those functionalities with a graphical interface, i.e. the Edit and Editor code provide with a simple (tree-like) editor for handling models conforming to the Ecore metamodel used as starting point. To that end, Edit and Editor code uses the Model code. Figure 5-14 shows an overview of this generation process.

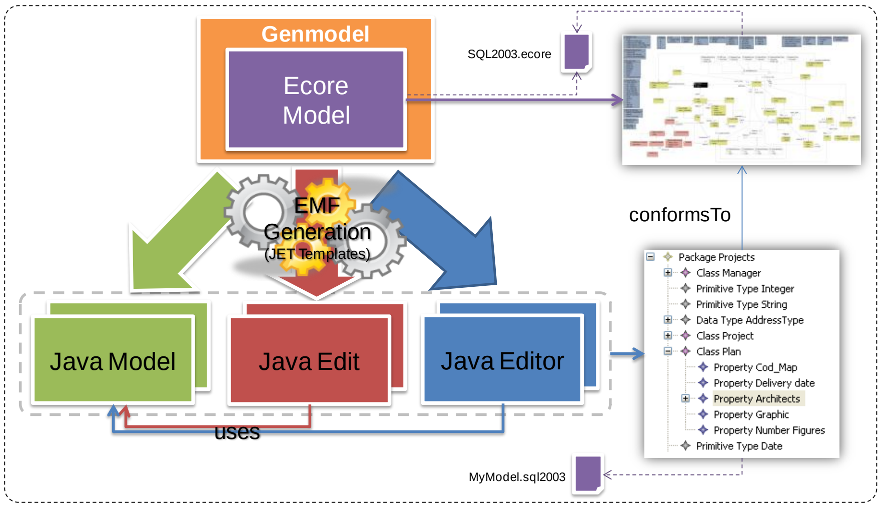
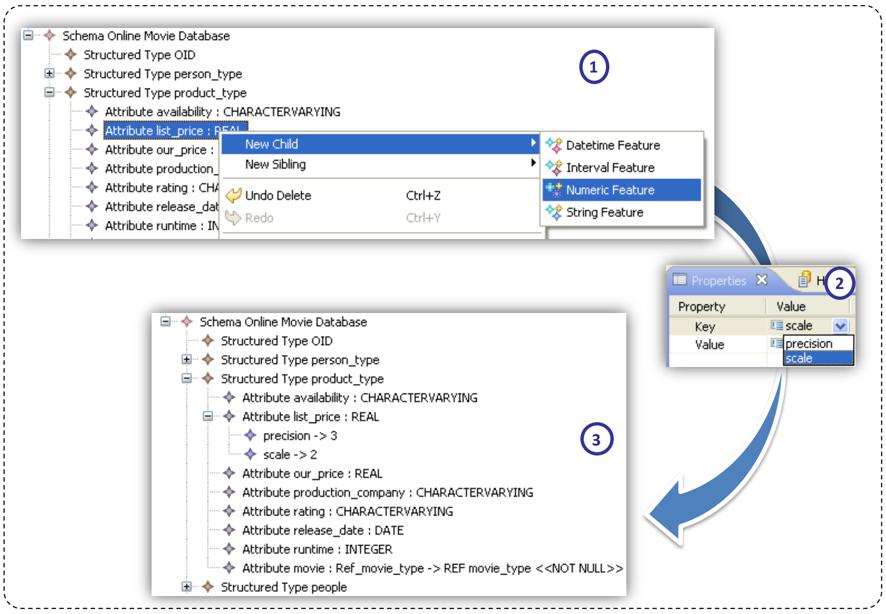


Figure 5-14. Overview of EMF Editors generation

From the .Ecore model (more properly, from that and the .genmodel) that collects the abstract syntax of the metamodel, the JAVA code that implements a simple, yet powerful, tree-like editor for conforming models is generated. This way, we can edit .sql2003 models conforming to the SQL2003 metamodel.

To conclude this section, Figure 2-5 shows show another example of using EMF editors. In particular, it focuses on how the metaclasses defined to support the modelling of built-in data types are to be instantiated in the EMF editor.

- First, we add a feature object nested on the *list_price* attribute. In particular, since we want the type of the attribute to be numeric, we create *Numeric Feature* object. (1)
- Next, we set the concrete feature to use, among the allowed features for number (*precision* and *scale*). To that end, we use the properties view of the tree-like editor. (2)
- Finally, see that the editor displays the *list_price* attribute. Whose type is REAL and whose size and scale has been also fixed to 2 and 3 respectively (3). Note that this model contains all the information needed to generate the SQL code that implements the designed schema.



```
Schema Online Movie Database
  Structured Type OID
  Structured Type person_type
  Structured Type product_type
    Attribute availability : CHARACTERVARYING
    Attribute list_price : REAL
    Attribute our_price :
    Attribute production_
    Attribute rating : CHA
    Attribute release_dat
    Attribute runtime : IN
```

```
Schema Online Movie Database
  Structured Type OID
  Structured Type person_type
  Structured Type product_type
    Attribute availability : CHARACTERVARYING
    Attribute list_price : REAL
      precision -> 3
      scale -> 2
    Attribute our_price : REAL
    Attribute production_company : CHARACTERVARYING
    Attribute rating : CHARACTERVARYING
    Attribute release_date : DATE
    Attribute runtime : INTEGER
    Attribute movie : Ref_movie_type -> REF movie_type <<NOT NULL>>
  Structured Type people
```

Figure 5-15. Using features on EMF editors

5.2.2.2 Customizing EMF editors:

The huge collection of primitive types supported by platform specific models, like the one from Oracle or the SQL standard hampers the definition of models using the tree-like editors of EMF. In general, all of them have to do with usability issues. One of the main concerns with MDE tools so far [318].

As well, we have already argued in favour of EMF tree-like editors over graphical editors for development tasks (see section 4.3.3.3). Nevertheless, the generic nature of EMF makes the generated editors too generic. Therefore we have worked to identify the way of adopting them to specific needs.

In the following, we present some results on the tree-like editors of M2DAT-DB. Note that the techniques applied will be also applied to develop the editors for the rest of DSLs that will integrate M2DAT.

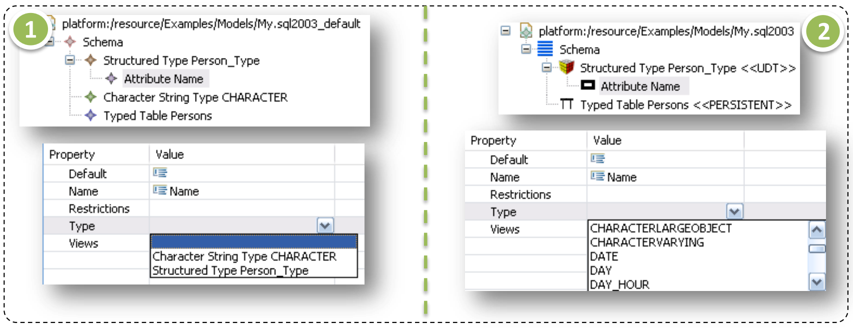
Including primitive types in any new model

When defining a PSM, each primitive type supported by the targeted platform has to be added manually in order to use it to define the type of any object in the model. That is, if the user wants an object to be of a particular type, he needs first to instantiate the metaclass that abstracts the type on the corresponding metamodel. For instance, back to the ORDB SQL:2003 DSL, if the user wants an attribute to be of type CHARACTER VARYING, he has to instantiate the *Character String Type* metaclass and sets its *descriptor attribute* to CHARACTER VARYING. Just think on the work needed to have at one's disposal the whole set of built-in types at the time of model edition.

We have identified the way to modify EMF (both the tree-like and the diagrammer) editors to overcome this issue. The technique provides with the following functionality: whenever a new PSM is created (in this case, an ORDB model for SQL:2003), the set of primitive types supported by that platform are automatically instantiated in the model. This way, when the user needs to assign a type to any of the objects in the model, he can use any of the built-in types supported by the targeted platform. To that end, we have modified the way new models are created in EMF.

Remember that a mandatory feature of any EMF model, because of EMF underlying XML format, is including a root element. Indeed, whenever a new model is created, a root object has to be created by selecting one of the metaclasses collected in the respective metamodel. We have followed the same approach to modify EMF generated code to bundle the built-in types in any new model.

Left-hand side of Figure 5-16 shows a screen capture from the EMF "default" editor from the SQL:2003 ORDB DSL, whereas right-hand side shows one from the M2DAT customized editor for the very same DSL.



platform:/resource/Examples/Models/My.sql2003_default
Schema
Structured Type Person_Type
Attribute Name
Character String Type CHARACTER
Typed Table Persons

Property	Value
Default	
Name	Name
Restrictions	
Type	
Views	

Character String Type CHARACTER
Structured Type Person_Type

platform:/resource/Examples/Models/My.sql2003
Schema
Structured Type Person_Type <<UDT>>
Attribute Name
Typed Table Persons <<PERSISTENT>>

Property	Value
Default	
Name	Name
Restrictions	
Type	
Views	

CHARACTERLARGEOBJECT
CHARACTERVARYING
DATE
DAY
DAY_HOUR

Figure 5-16. Assigning primitive types in EMF "Default" editor VS M2DAT EMF Editor

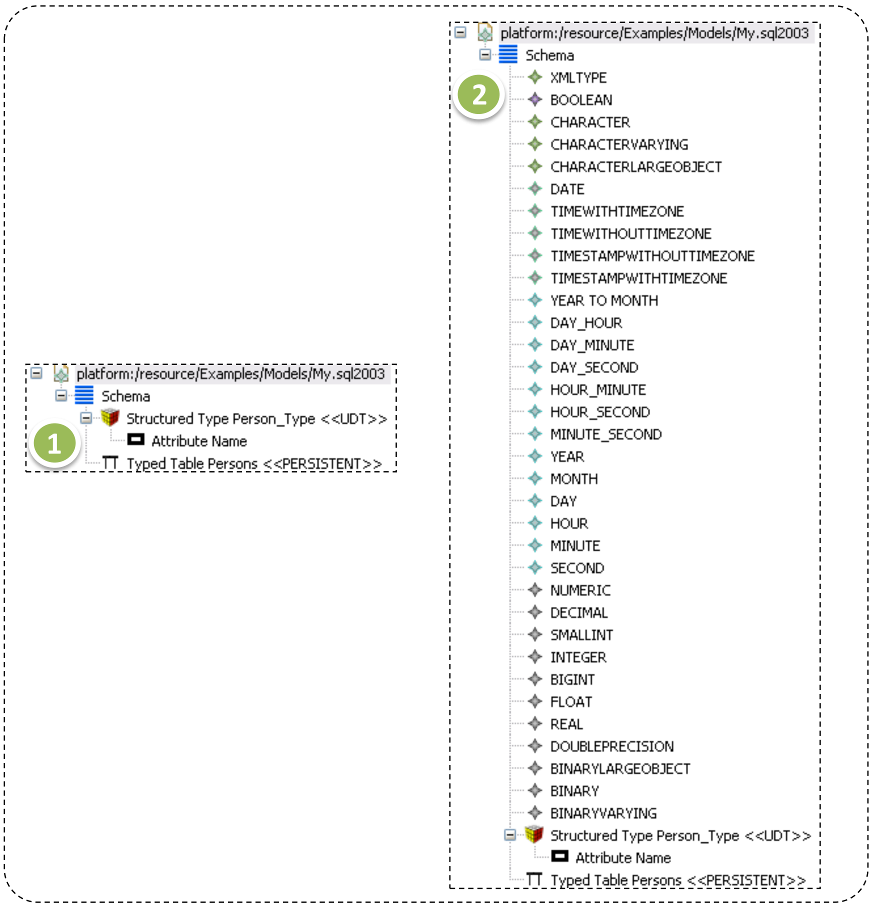
In both cases, the objective is to define an attribute *Name* in *Person_Type* and assign it a character type. In the first case (1), the *Character String Type* object has to be created. In addition, notice just the newly created type and the *Person_Type* can be used to define the type of the attribute, i.e. just those objects visualized in the editor. In contrast, when using M2DAT modified editor (2), any of the SQL:2003 built-in types could be used at the time of defining the type of the attribute since they were already instantiated when the model was created.

Finally, we would like to mention that the same technique is applied for GMF-graphical editors.

Hiding primitive types in EMF editors

Previous section showed the customization of EMF tree-like editors to include the built-in types of the corresponding DSL in any newly created model. However, if we limit to include them in the model, we are adding too much "noise". Indeed, when a new model is edited (using the diagrammer or the tree-like editor) it contains a huge number of static objects, i.e. objects that will not be modified. They are needed just to define the type of new elements to add in the model. Apart from that task, they just serve to add distraction. In other words, displaying all the (already created) primitive types in the editor acts against usability.

Figure 5-17 compares the effect of filtering the objects corresponding to the already instantiated primitive types (1) versus a non-filtered view of the model (2). Notice that the functionality provided is exactly the same: any SQL:2003 primitive type can be used to define the type of a model element. Nevertheless, they differ visibly regarding usability.



```
platform:/resource/Examples/Models/My.sql2003
  Schema
    Structured Type Person_Type <<UDT>>
      Attribute Name
    Typed Table Persons <<PERSISTENT>>
```

```
platform:/resource/Examples/Models/My.sql2003
  Schema
    XMLTYPE
    BOOLEAN
    CHARACTER
    CHARACTERVARYING
    CHARACTERLARGEOBJECT
    DATE
    TIMEWITHTIMEZONE
    TIMEWITHOUTTIMEZONE
    TIMESTAMPWITHOUTTIMEZONE
    TIMESTAMPWITHTIMEZONE
    YEAR TO MONTH
    DAY_HOUR
    DAY_MINUTE
    DAY_SECOND
    HOUR_MINUTE
    HOUR_SECOND
    MINUTE_SECOND
    YEAR
    MONTH
    DAY
    HOUR
    MINUTE
    SECOND
    NUMERIC
    DECIMAL
    SMALLINT
    INTEGER
    BIGINT
    FLOAT
    REAL
    DOUBLEPRECISION
    BINARYLARGEOBJECT
    BINARY
    BINARYVARYING
    Structured Type Person_Type <<UDT>>
      Attribute Name
    Typed Table Persons <<PERSISTENT>>
```

Figure 5-17. Filtering instantiated primitive types in M2DAT editors

Filtering elements to be added on a model

The last issue related with the huge amount of primitive types supported by any “real” platform refers to the creation of new elements in a given model.

Whenever the user wants to add a new element, he clicks on the menu “Create Child”. Then, a combo box is open out showing all the metaclasses, i.e. all the classes included in the corresponding metamodel. This way, he can instantiate the one he needs. However, including all the metaclasses that serve to capture primitive types hampers usability. Given that all the supported primitive types are already instantiated at model creation, there is no need to allow the user creating new primitive type objects.

Once again, we modify EMF generated editors to solve this drawback. All the metaclasses corresponding to primitive types, i.e. all the metaclasses that inherits from Predefined Type in the case of SQL:2003 metamodel (see Figure 5-2), are automatically filtered in the combo box used to add new elements to a given model. Figure 5-18 shows a screen capture from the M2DAT modified editor (1) and the default one where no filter is applied (2).

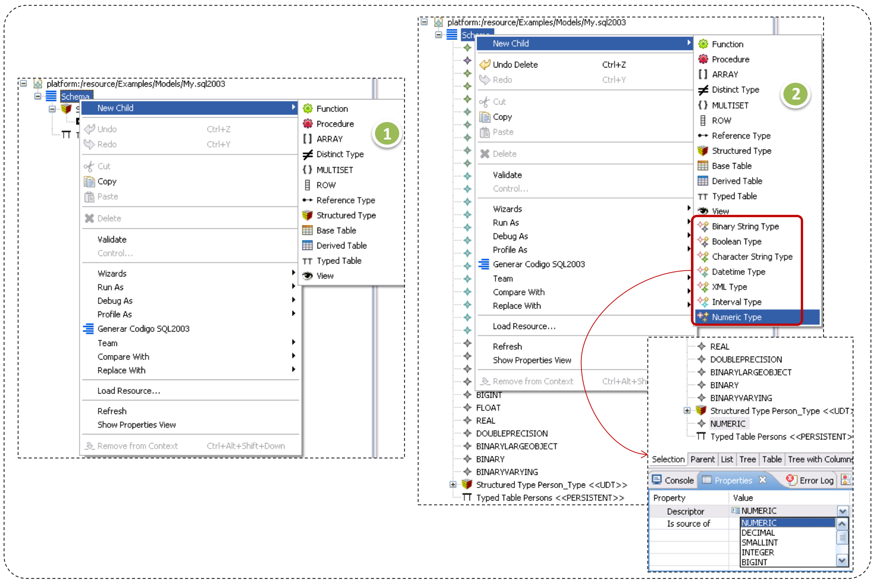


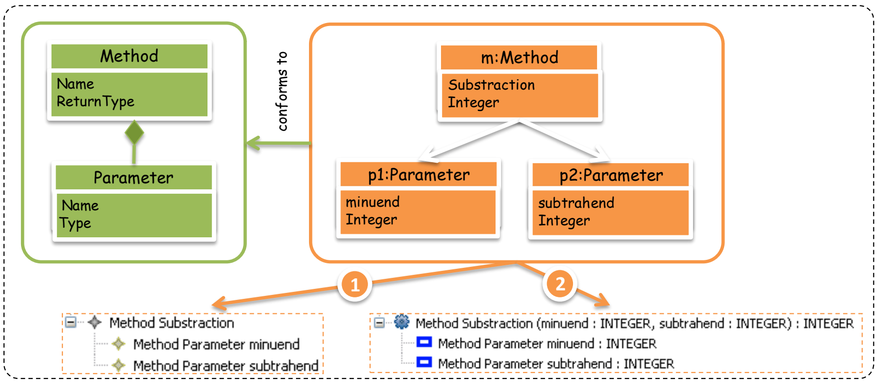
Figure 5-18. Filtering metaclasses to instantiate in M2DAT editors

In addition, the figure serves to illustrate the effect of not preventing the creation of new Primitive Types. After instantiating the selected family (Numeric in the picture), the user must select which one from the concrete types of this family he deserves to instantiate, i.e. DECIMAL, SMALLINT, INTEGER, BIGINT, etc.

Enhancing user feedback on M2DAT Editors

Another improvement we would like to comment on is related with the way information about each model element is displayed in the editors. To introduce the problem and how it is solved we use a very simple example shown in Figure 5-19.

Left-hand side of the picture shows a simple metamodel to model methods and its parameters while right-hand side shows a sample instantiation. The *Subtraction* method receives two parameters: *p1* and *p2*. Both are Integers and respectively the minuend and subtraend of the difference. The bottom of Figure 5-19 shows how this method is displayed in the tree-like editor generated by EMF (1) and the M2DAT improved editor (2). The latter shows not only the name of the method, but also which parameters it receives plus the type of each one.



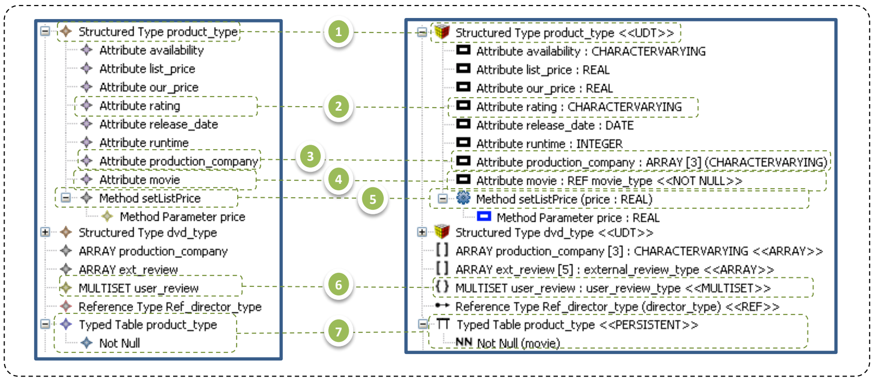
Method Substraction
Method Parameter minuend
Method Parameter subtrahend

Method Substraction (minuend : INTEGER, subtrahend : INTEGER) : INTEGER
Method Parameter minuend : INTEGER
Method Parameter subtrahend : INTEGER

Figure 5-19. Displaying a method signature on EMF editors

To than end, both the JAVA code generated by EMF to display *Method* objects and *Parameter* objects has been modified. We have added a *getFriendlyName* to the JAVA interfaces generated for each metaclass of the starting metamodel. Likewise, we have redefined the *getText()* method to invoke *getFriendlyName()*. No need to say, the information provided by the modified editor is much better in terms of usability.

We have applied the same principle to modify M2DAT editors in order to enhance their usability. To illustrate the result Figure 5-20 shows the OMDB model used as a case study so far displayed in the EMF “default” editor and the M2DAT improved one.



Structured Type product_type
Attribute availability
Attribute list_price
Attribute our_price
Attribute rating
Attribute release_date
Attribute runtime
Attribute production_company
Attribute movie
Method setListPrice
Method Parameter price
Structured Type dvd_type
ARRAY production_company
ARRAY ext_review
MULTISET user_review
Reference Type Ref_director_type
Typed Table product_type
Not Null

Structured Type product_type <<UDT>>
Attribute availability : CHARACTERVARYING
Attribute list_price : REAL
Attribute our_price : REAL
Attribute rating : CHARACTERVARYING
Attribute release_date : DATE
Attribute runtime : INTEGER
Attribute production_company : ARRAY [3] (CHARACTERVARYING)
Attribute movie : REF movie_type <<NOT NULL>>
Method setListPrice (price : REAL)
Method Parameter price : REAL
Structured Type dvd_type <<UDT>>
ARRAY production_company [3] : CHARACTERVARYING <<ARRAY>>
ARRAY ext_review [5] : external_review_type <<ARRAY>>
MULTISET user_review : user_review_type <<MULTISET>>
Reference Type Ref_director_type (director_type) <<REF>>
Typed Table product_type <<PERSISTENT>>
Not Null (movie)

Figure 5-20. EMF Default editor VS M2DAT improved editor: OMDB for SQL:2003 model

First of all, we have already mentioned in this dissertation our inclination in favour of DSLs with a UML-like flavour (see section 4.2.3). That is, models that looks as UML profiles, to take advantage from the universal nature of UML. Actually, they are defined with a DSL to ease the task of processing them. In line with this idea, M2DAT editors are modified to show, next to any modelled element, the visual stereotype assigned to the corresponding metaclass. This way, we ease the task of identifying any desired element in the model.

Back to the figure, the name of each structured type, like the *product_type* (1) is followed by its corresponding stereotype (<<UDT>>). Besides, its attribute displays, not only its name, but also its type (2). Even if the attribute's type is composed, like the *production company* attribute, the type is described completely (3). If any restriction has been defined over the attribute, it is also shown in-line (4). As well, when it is a Reference type, the type referenced is shown (4). Next to each method name, the name and type of its parameters is displayed in-line (5), i.e. the complete signature. Collection types are described displaying the type of each item and the corresponding stereotype (6). Finally, next to the restrictions defined over each table, the attributes/columns affected by the constraint are also displayed (7).

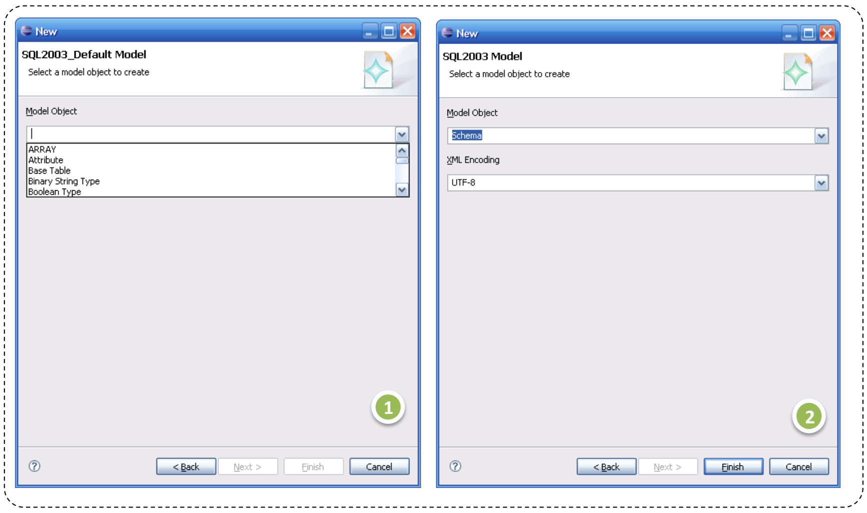
Automatic Identification of root elements

We would like to comment a last usability improvement on M2DAT editors. We have already mentioned that any EMF model has to include a root element. Thus, whenever a new model is created the corresponding wizard asks the user to select a metaclass to be instantiated as the root element. When the metamodel is large enough, finding the right metaclass might be annoying. To avoid the need for such selection, we have modified EMF generated code. This way, the wizard will identify automatically the root element when a new model is created. Figure 5-21 shows the original wizard (1) versus the improved one integrated n M2DAT (2).

Ongoing work

Finally, it is worth mentioning that we plan to integrate all these modifications in EMF itself. That is, instead of modifying the generated editors, we are studying the way to modify EMF generation process in order to include all these capabilities in any newly created EMF-based editor.

To that end, since EMF is migrating GMF code generation to XPand, we have already started to use XPand as a model-to-text transformation language to develop other M2DAT modules. The aims is at mastering Xpand to be able to adapt EMF's Xpand templates to our needs.



Model Object: Schema

XML Encoding: UTF-8

Figure 5-21. Setting root element in EMF editors

5.2.2.3 GMF Implementation

We already gave an overview of the development process for GMF editors in section 4.3.2. Therefore, here we will focus just on how its is used in the framework of M2DAT. To that end we show its application to develop the SQL:2003 diagrammer.

The idea is summarized in Figure 5-22: two GMF models, the Graphical model and the Tooling model collect the graphical information (the concrete syntax) for the new DSL. The mapping between the concrete syntax and the abstract syntax is depicted in another model, the Mapping model. Then, a Generator model is automatically obtained. As well as with EMF generation, the generator model encodes some details to drive the generation process. Finally, the JAVA code that implements the editor, i.e. the Diagram(mer) plug-in is generated. From there on, the user can edit .sql2003 models diagramatically. To that purpose, for every .sql2003 model, an .sql2003_diagram is created. The later contains the data related with the visual presentation of the model (its concrete syntax), while the model itself (its abstract syntax) remains in the .sql2003 model.

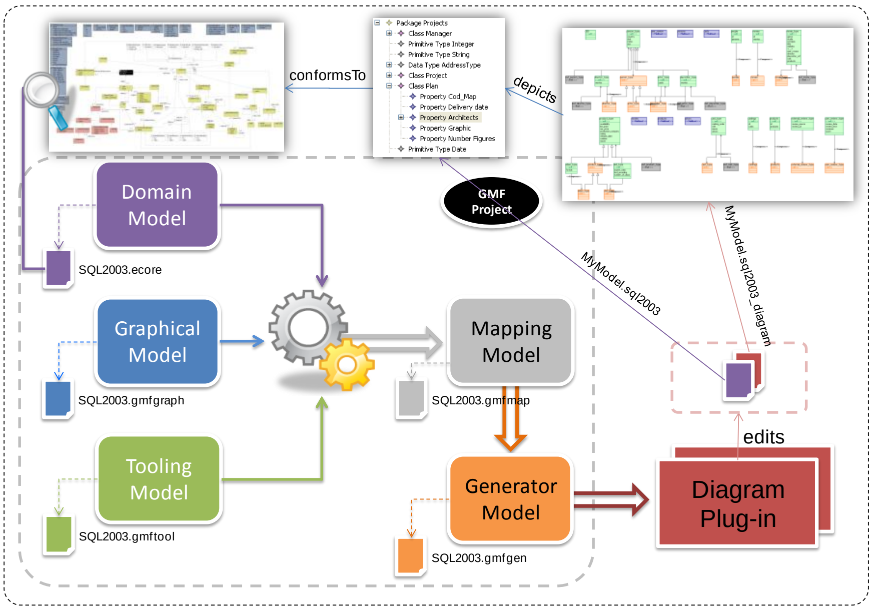
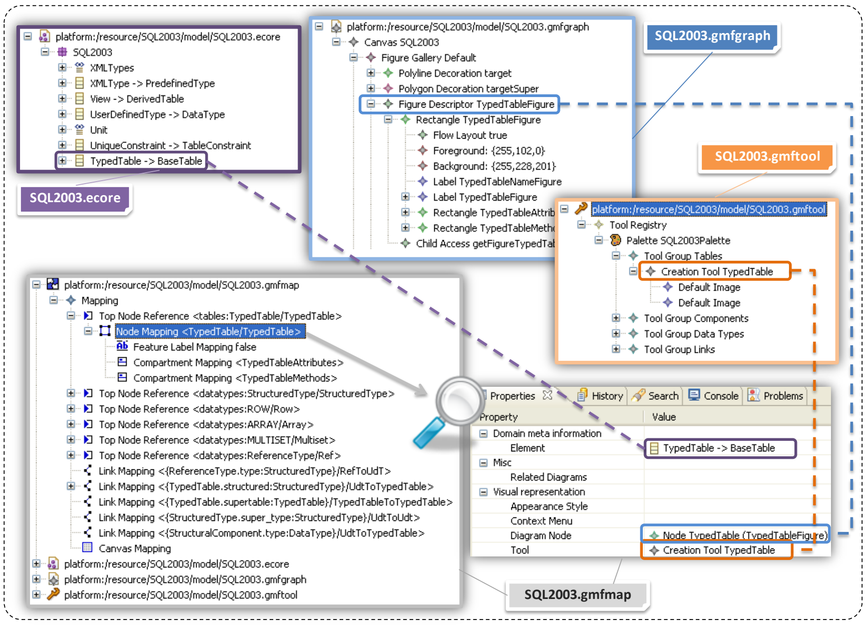


Figure 5-22. GMF Overview

GMF is a perfect example of model-driven development since the generation of a GMF graphical editor is driven by a set of models. In the following, we will present each of them using the case study we have followed so far, the development of the diagrammer for ORDB SQL:2003 models. We will focus on the specification of how Typed Tables should be represented. To that purpose, Figure 5-23 shows partial views of the GMF models used. In particular, those parts referring to the representation of Typed Tables have been bordered with coloured rectangles.



```
platform:/resource/SQL2003/model/SQL2003.ecore
  SQL2003
    XMLTypes
    XMLType -> PredefinedType
    View -> DerivedTable
    UserDefinedType -> DataType
    Unit
    UniqueConstraint -> TableConstraint
    TypedTable -> BaseTable
```

```
platform:/resource/SQL2003/model/SQL2003.gmfgraph
  Canvas SQL2003
    Figure Gallery Default
      Polyline Decoration target
      Polygon Decoration targetSuper
      Figure Descriptor TypedTableFigure
        Rectangle TypedTableFigure
          Flow Layout true
          Foreground: {255,102,0}
          Background: {255,228,201}
          Label TypedTableNameFigure
          Label TypedTableFigure
          Rectangle TypedTableAttrib
          Rectangle TypedTableMeth
        Child Access getFigureTypedTal
```

```
platform:/resource/SQL2003/model/SQL2003.gmftool
  Tool Registry
    Palette SQL2003Palette
      Tool Group Tables
        Creation Tool TypedTable
          Default Image
          Default Image
      Tool Group Components
      Tool Group Data Types
      Tool Group Links
```

```
platform:/resource/SQL2003/model/SQL2003.gmfmap
  Mapping
    Top Node Reference <tables:TypedTable/TypedTable>
      Node Mapping <TypedTable/TypedTable>
        Feature Label Mapping false
        Compartment Mapping <TypedTableAttributes>
        Compartment Mapping <TypedTableMethods>
    Top Node Reference <datatypes:StructuredType/StructuredType>
    Top Node Reference <datatypes:ROW/Row>
    Top Node Reference <datatypes:ARRAY/Array>
    Top Node Reference <datatypes:MULTISET/Multiset>
    Top Node Reference <datatypes:ReferenceType/Ref>
    Link Mapping <{ReferenceType.type:StructuredType}/RefToUdT>
    Link Mapping <{TypedTable.structured:StructuredType}/UdtToTypedTable>
    Link Mapping <{TypedTable.supertable:TypedTable}/TypedTableToTypedTable>
    Link Mapping <{StructuredType.super_type:StructuredType}/UdtToUdt>
    Link Mapping <{StructuralComponent.type:DataType}/UdtToTypedTable>
    Canvas Mapping
platform:/resource/SQL2003/model/SQL2003.ecore
platform:/resource/SQL2003/model/SQL2003.gmfgraph
platform:/resource/SQL2003/model/SQL2003.gmftool
```

```
Properties | History | Search | Console | Problems
Property                   Value
Domain meta information
    Element                TypedTable -> BaseTable
Misc
    Related Diagrams
Visual representation
    Appearance Style
    Context Menu
    Diagram Node           Node TypedTable (TypedTableFigure)
    Tool                   Creation Tool TypedTable
```

Figure 5-23. GMF models to develop the SQL:2003 Graphical Editor

First of all, the domain model collects the abstract syntax of the DSL. Actually, when we refer to the domain model, we are referring to the DSL metamodel (*SQL2003.Ecore*).

Next, all the graphical elements that will appear in the resulting editor are defined in the *SQL2003.gmfgraph* model. For instance, it includes a Figure Descriptor object called *TypedTableFigure*. This object collects all the graphical information needed to represent Typed Tables. Note that the figure is a Rectangle, whose foreground and background colours are fixed using a Foreground and Background nested objects. Besides, it contains two labels to show the name and the stereotype deserved. In addition, two more rectangles are nested to show the attributes and the methods of the Typed Table.

Besides, any diagrammer has to provide with controls to add new elements to the diagram. This way, the *SQL2003.gmftool* model specifies which controls will be included in the diagrammer. In particular, note the *TypedTable Creation Tool* object that will allow adding new Typed Table objects to a model.

At this moment, there is still no connection between the graphical elements specified in the graphical and tool models and the domain concepts collected in the domain model (the metamodel). The definition of these correspondences is done in the mapping model (*SQL2003.gmfmap*). If you look at the properties of the Node Mapping *TypedTable/TypedTable* you will find the links defined for representing Typed Tables. The node mapping links the domain element TypedTable (that inherits from BaseTable), with the diagram node TypedTable and the Creation Tool TypedTable.

Finally, the generator model is automatically obtained. It contains the options that drive the generation of the JAVA code that compose the plug-in implementing the diagrammer.

It is worth mentioning that GMF rests extensively on EMF generated code. How model elements are displayed on GMF editors, the icons used to identify them, even the labels that show their names, are directly taken from EMF generated code. Thus, the improvements over the EMF tree-like editor showed in section 5.2.2.2 are automatically transfered to the GMF editor.

To conclude this section, we would like to mention that we have provided here a very simplified version of the development of graphical editors in M2DAT. Indeed, working this way, a *default* editor is obtained at the end of the process. In some cases it could be enough, but if the editor is thought to be distributed, generated code should be probably modified in order to get the desired look and feel and behaviour. In this sense, it is also remarkable that GMF code is far from being trivial. This is due to the fact that, as it happens with UML, the objective of having a one-size-fits-all solution results in too much complexity.

5.3 Model Transformations in M2DAT

Several times along this dissertation we have stressed the role of model transformations in MDE development processes. They are the key to automate and drive the process. Therefore, we will show how model transformations are implemented in M2DAT using the solutions selected. Those that were introduced in the previous chapter.

To that purpose, we indentify a set of common generic scenarios to address when developing model transformations. Each scenario is defined by the set of constructions that compose the source and target pattern in each case. For instance, one common scenario is the following: the existence of one element in the source model implies the creation of several elements in the target model.

For each scenario we will show how it is implemented using ATL and, when a design decision is needed, AMW for collecting such decision. Afterwards, we will explain a number ok key issues and lessons learned.

5.3.1 Common Scenarios

It is not our intention to show all the transformations coded during the development of M2DAT-DB, but provide with a set of common scenarios found when developing model transformations and show how they are addressed. The underlying ideas are:

- On the one hand, to identify the techniques or strategies used to address each of these scenarios. They will be applicable in the transformations to develop in forthcoming M2DAT prototypes.
- On the other hand, to prove that the components and techniques used to develop model transformations in M2DAT are valid to address any possible scenario. This is achieved to agreat extent through the use of annotation models to drive model transformation executions.

Table 5-1 summarizes the common scenarios identified in model-to-model transformations. We distinguish them according to the number of source elements in the source pattern (1 – N) and the number of elements of the target pattern (1 – N). Besides, we make a difference between those cases in which the source pattern is always mapped to the same target pattern (FIXED), and those in which different target patterns could be instantiated to map the matched source pattern (OPTIONAL). The latter needs from a design decision to state which target pattern is to be used.

Table 5-1. Common Scenarios for Model-to-Model transformations

TARGET MODEL / SOURCE MODEL	1		N	
	FIXED	OPTIONAL	FIXED	OPTIONAL
1	X	X	X	X
N	X	X	X	X

In following subsections we show an example of occurrence of each scenario, next to how we have addressed its implementation in the UML2SQL2003 transformation embedded in M2DAT-DB. As explained in section 5.1.1, this transformation generates ORDB models conforming to SQL:2003 from a pure conceptual data model depicted in a UML class diagram. As well, we will show the application of the rules using excerpts from the Case Study used so far, the Online Movie Database (remember that the whole Case Study can be found in Appendix D).

5.3.1.1 One–to–One

This scenario refers to those situations in which there is a one-to-one correspondence between a metaclass from the source metamodel and another from the target one. It is sketched in Figure 5-24. Obviously, this is the simplest situation to address and we found many examples in any model transformation. Indeed, we should aim at expressing all the rules in this way in order to keep simple the transformation and ease the maintenance of traceability links. However, this is just feasible for quite simple metamodels or at least those that are semantically closer.

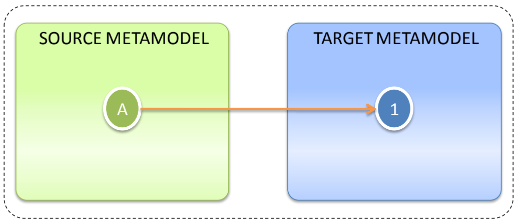
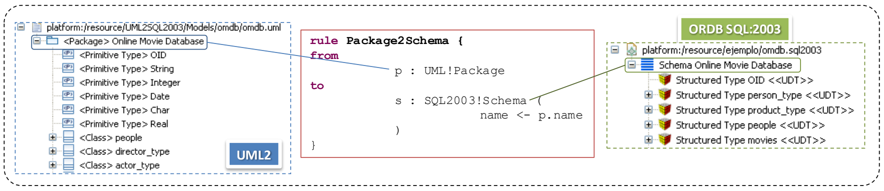


Figure 5-24. One-to-One transformation

In the UML2SQL2003 model transformation we can find a number of rules that tackle this type of scenario. For instance, since we have already mentioned that every Ecore metamodel has to own a root element, we need something akin to a "root" rule to map them. This is a very simple rule shown in Figure 5-25.



```
platform:/resource/UML2SQL2003/Models/omdb/omdb.uml
  <Package> Online Movie Database
    <Primitive Type> OID
    <Primitive Type> String
    <Primitive Type> Integer
    <Primitive Type> Date
    <Primitive Type> Char
    <Primitive Type> Real
    <Class> people
    <Class> director_type
    <Class> actor_type
```

```
rule Package2Schema {
from
        p : UML!Package
to
        s : SQL2003!Schema (
                name <- p.name
        )
}
```

```
platform:/resource/ejemplo/omdb.sql2003
  Schema Online Movie Database
    Structured Type OID <<UDT>>
    Structured Type person_type <<UDT>>
    Structured Type product_type <<UDT>>
    Structured Type people <<UDT>>
    Structured Type movies <<UDT>>
```

Figure 5-25. ATL Rule Package2Schema

The source pattern states that the rule will match any Package found on the UML source model. The target pattern states that for each match, i.e. for every Package, a Schema is created in the target model. Besides, the name of the newly created Schema will be that of the matched Package (*Online Movie Database* in the Case Study).

5.3.1.2 One–to–Many

This situation is a little bit more complex that the previous one, but still almost trivial. As Figure 5-26 illustrates, this time the source pattern contains just one element while the target pattern contains several elements.

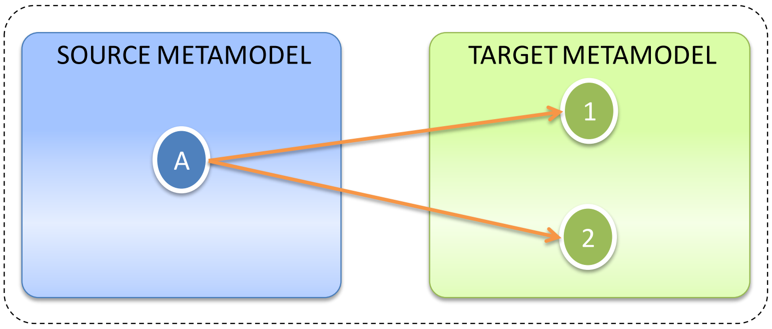
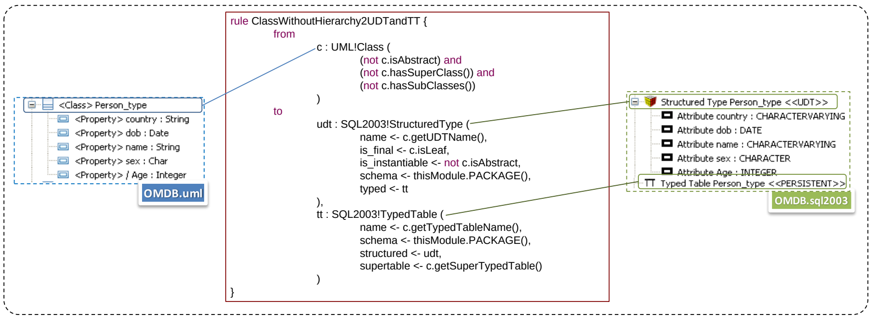


Figure 5-26. One-to-Many transformation

Back to the UML2SQL2003 transformation, a generic rule states that every Class from the conceptual data model is mapped to an Structured Type (so-called UDT from now on) plus a Typed Table. Please, note that this rule applies just for the generic case, that will be later refined attending to the nature of each artefact. For instance, if the class is the parent class in some generalization, the mapping might not be direct.

To that end, the rule *ClassWithoutHierarchy2UDTandTT* (Figure 5-27) includes a guard that ensures that only instantiable UML classes that do not participate in any hierarchy will match this rule. For each matched class, an Structured Type and a Typed Table (whose type is the newly created) are added to the target model.



```
rule ClassWithoutHierarchy2UDTandTT {
        from
                c : UML!Class (
                        (not c.isAbstract) and
                        (not c.hasSuperClass()) and
                        (not c.hasSubClasses())
                )
        to
                udt : SQL2003!StructuredType (
                        name <- c.getUDTName(),
                        is_final <- c.isLeaf,
                        is_instantiable <- not c.isAbstract,
                        schema <- thisModule.PACKAGE(),
                        typed <- tt
                ),
                tt : SQL2003!TypedTable (
                        name <- c.getTypedTableName(),
                        schema <- thisModule.PACKAGE(),
                        structured <- udt,
                        supertable <- c.getSuperTypedTable()
                )
}
```

```
<Class> Person_type
    <Property> country : String
    <Property> dob : Date
    <Property> name : String
    <Property> sex : Char
    <Property> / Age : Integer
OMDB.uml
```

```
Structured Type Person_type <<UDT>>
    Attribute country : CHARACTERVARYING
    Attribute dob : DATE
    Attribute name : CHARACTERVARYING
    Attribute sex : CHARACTER
    Attribute Age : INTEGER
TT Typed Table Person_type <<PERSISTENT>>
OMDB.sql2003
```

Figure 5-27. ATL Rule ClassWithoutHierarchy2UDTandTT

Note also that this refers only to the mapping of the class. Its attributes and methods are handled as isolated objects that will be mapped by other rules. Indeed, this is the main advantage of adopting declarative approaches (actually, hybrid with emphasis on the declarative style): when implementing the mapping of one element, there is no need to worry about how related elements are mapped. The underlying engine ensures that they will be mapped.

5.3.1.3 Many–to–One

This scenario is depicted in Figure 5-28. Two elements from the source metamodel has to be mapped to the same element in the target metamodel.

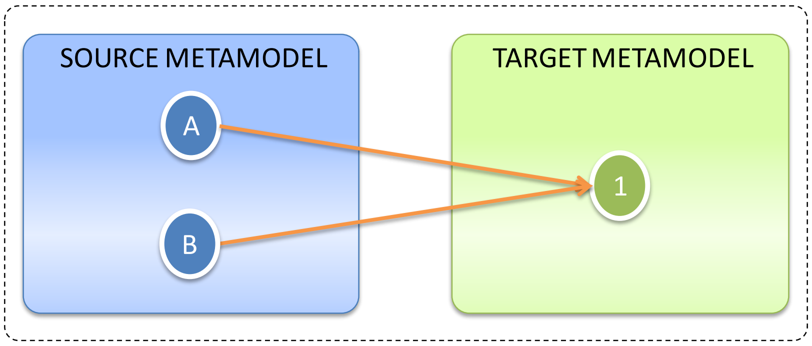
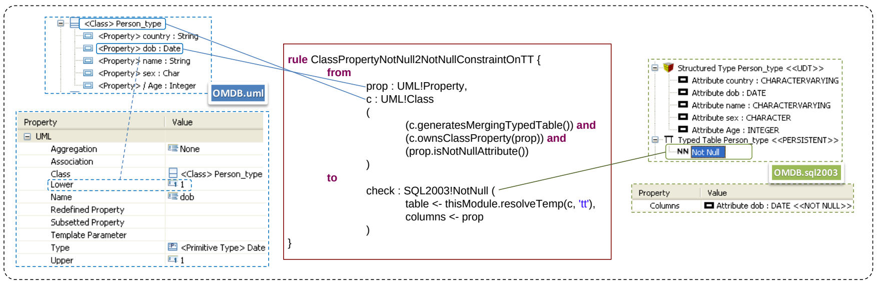


Figure 5-28. Many-to-One transformation

Although this scenario seems to be as simple as the previous one, it is much more challenging. In fact, a declarative approach implies that for each element to match in the source model, one element (or more) have to be created in the target model, i.e. declarative approaches implement injective transformations. Nevertheless we do not want to implement an injective function, but a surjective one, where the occurrence of a set of elements in the source model induces the creation of just one element in the target model.

Hopefully, the improvements on the last version of ATL engine (ATL-VM 2006), in particular the support for defining rules with multiple source patterns, simplifies the implementation of this scenario. This way, we apply such ATL feature to implement the mapping of cardinalities from conceptual models to ORDB models. For instance, if there is an UML property whose multiplicity lower bound is 1 or greater, we have to control that the corresponding OR attribute does not take a null value. To that end, we have to add a Not Null constraint on every table defined over the Structured Type that contains such attribute. An example is shown in Figure 5-29, next to the ATL rule that encodes its management.



```
<Class> Person_type
    <Property> country : String
    <Property> dob : Date
    <Property> name : String
    <Property> sex : Char
    <Property> / Age : Integer
```

OMDB.uml

Property	Value
UML	
Aggregation	None
Association	
Class	<Class> Person_type
Lower	1
Name	dob
Redefined Property	
Subsetted Property	
Template Parameter	
Type	<Primitive Type> Date
Upper	1

```
rule ClassPropertyNotNull2NotNullConstraintOnTT {
        from
                prop : UML!Property,
                c : UML!Class
                (
                        (c.generatesMergingTypedTable()) and
                        (c.ownsClassProperty(prop)) and
                        (prop.isNotNullAttribute())
                )
        to
                check : SQL2003!NotNull (
                        table <- thisModule.resolveTemp(c, 'tt'),
                        columns <- prop
                )
}
```

```
Structured Type Person_type <<UDT>>
    Attribute country : CHARACTERVARYING
    Attribute dob : DATE
    Attribute name : CHARACTERVARYING
    Attribute sex : CHARACTER
    Attribute Age : INTEGER
TT Typed Table Person_type <<PERSISTENT>>
    NN Not Null
```

OMDB.sql2003

Property	Value
Columns	Attribute dob : DATE <<NOT NULL>>

Figure 5-29. ATL Rule ClassPropertyNotNull2NotNullConstraintOnTT

The *Person_type* Class owns a *dob* Property, whose multiplicity lower bound is 1. The union of the Class and the Property matches the source pattern defined in the ATL rule. As well, the guard restricts the possible matches by allowing just classes that do generate a Typed Table (since abstract classes mapping do not generate a Typed Table) and Properties that have to be mapped with Not Null constraints (the *isNotNullAttribute* helper ensures this). Since there is a positive matching, a Not Null constraint is added to the target model. It is defined over the *Person_type* Typed Table and refers to the *dob* attribute.

5.3.1.4 Many-to-Many

Next, we focus on the scenario sketched in Figure 5-30.

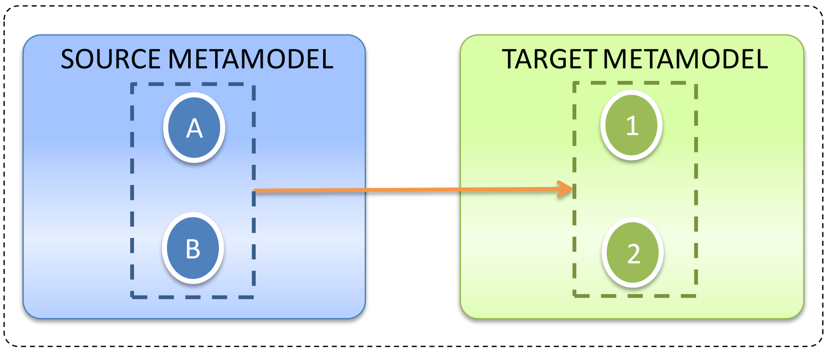


Figure 5-30. Many-to-Many transformation

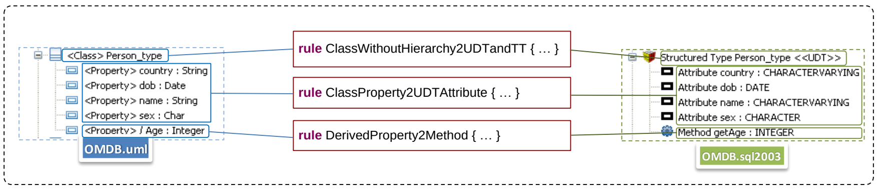
We can look at this case in two different ways:

- As a variation of the previous one: there is still several elements in the construction to find in the source model, but there is a number of elements in the construction to create on the target model. Though we have not implemented this scenario as-is in M2DAT-DB so far, it is not complex. Indeed, the ATL rule from the previous case with some minor modifications could be used to implement this case. For instance, the rule from Figure 5-31 serves to implement the situation shown in Figure 5-30

```
rule AB_2_12 {
        from
                a : SourceMM!A,
                b : SourceMM!B
                (
                        a.checkSomeStuff() and
                        b.chekSomeOtherStuff()
                )
        to
                one : TargetMM!One (
                        my_property <- a.property
                ),
                two : TargetMM!Two (
                        my_property <- b.property
                )
}
```

Figure 5-31. ATL Rule Many-to-Many (Generic)

- As a composition of more simple situations. This is the case of a Class and its Properties mapped to a Structured Type and the corresponding attributes. We illustrate the situation in Figure 5-32 with the mapping of the *Person_type* class.



<Class> Person_type
<Property> country : String
<Property> dob : Date
<Property> name : String
<Property> sex : Char
<Property> / Age : Integer
OMDB.uml
rule ClassWithoutHierarchy2UDTandTT { … }
rule ClassProperty2UDTAttribute { … }
rule DerivedProperty2Method { … }
Structured Type Person_type <<UDT>>
Attribute country : CHARACTERVARYING
Attribute dob : DATE
Attribute name : CHARACTERVARYING
Attribute sex : CHARACTER
Method getAge : INTEGER
OMDB.sql2003

Figure 5-32. Many-to-Many transformation decomposed into One-to-One transformations

The mapping of each element from the source pattern (i.e. the class and each property) is carried out by a different rule. This way, the *Person_Type* Class is mapped by the *ClassWithoutHierarchyToUDTandTT* rule, the Primitive type Properties (*country*, *dob*, *name* and *sex*) are mapped by means of the *ClassProperty2UDTAttribute* rule and the Derived Property (*Age*) is mapped by the *DerivedProperty2Method* rule.

5.3.1.5 One-to-One (multiple options)

As illustrated in Figure 5-33, this scenario differs from the previous one in the sense that the source pattern admits two possible target patterns, i.e. the A object from the source model may be mapped as an object of class 1 or as an object of class 1'.

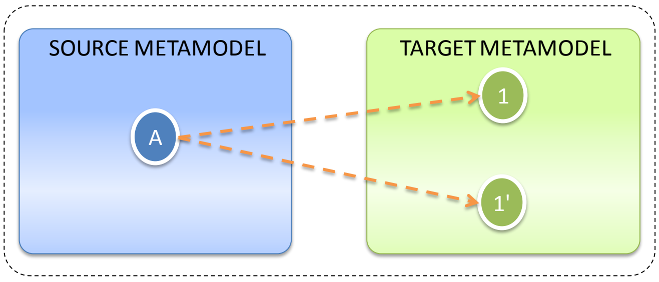
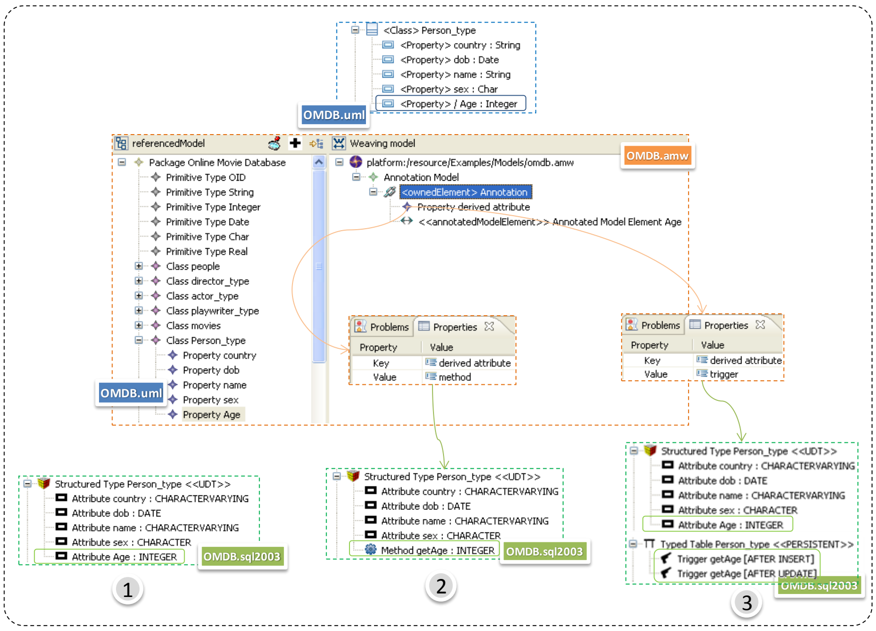


Figure 5-33. One-to-One transformation (multiple options)

Following with the UML2SQL2003 transformation, a generic rule states that every Property of a Class is to be mapped as an attribute in the corresponding Structured Type.

Nevertheless, derived attributes admits two extra ways of mapping: as a method or as a simple attribute plus a trigger to calculate its value. We can annotate the source model to discern which rule has to be applied for a given matching, i.e. for a particular derived property. This way, Figure 5-34 illustrates the three possibilities.



```
<Class> Person_type
    <Property> country : String
    <Property> dob : Date
    <Property> name : String
    <Property> sex : Char
    <Property> / Age : Integer
```

OMDB.uml

```
referencedModel
Package Online Movie Database
    Primitive Type OID
    Primitive Type String
    Primitive Type Integer
    Primitive Type Date
    Primitive Type Char
    Primitive Type Real
    Class people
    Class director_type
    Class actor_type
    Class playwriter_type
    Class movies
    Class Person_type
        Property country
        Property dob
        Property name
        Property sex
        Property Age
```

OMDB.uml

```
Weaving model
platform:/resource/Examples/Models/omdb.amw
    Annotation Model
        <ownedElement> Annotation
            Property derived attribute
            <<annotatedModelElement>> Annotated Model Element Age
```

OMDB.amw

Property	Value
Key	derived attribute
Value	method

Property	Value
Key	derived attribute
Value	trigger

```
Structured Type Person_type <<UDT>>
    Attribute country : CHARACTERVARYING
    Attribute dob : DATE
    Attribute name : CHARACTERVARYING
    Attribute sex : CHARACTER
    Attribute Age : INTEGER
```

OMDB.sql2003

1

```
Structured Type Person_type <<UDT>>
    Attribute country : CHARACTERVARYING
    Attribute dob : DATE
    Attribute name : CHARACTERVARYING
    Attribute sex : CHARACTER
    Method getAge : INTEGER
```

OMDB.sql2003

2

```
Structured Type Person_type <<UDT>>
    Attribute country : CHARACTERVARYING
    Attribute dob : DATE
    Attribute name : CHARACTERVARYING
    Attribute sex : CHARACTER
    Attribute Age : INTEGER
Typed Table Person_type <<PERSISTENT>>
    Trigger getAge [AFTER INSERT]
    Trigger getAge [AFTER UPDATE]
```

OMDB.sql2003

3

Figure 5-34. Different ways of mapping derived attributes

In the first case (1), we do not annotate the *Age* Property of the *Person_type* class (the slash preceding the name of the property denotes that it is a derived property according to UML). Thus, it is mapped by adding an *Age* attribute on the *Person_Type* structured type.

In the second case (2), we annotates the Age property. To that end, we add an Annotation object in the corresponding weaving model (*OMDB.amw*). As we described in section 4.8.4.2, each annotation contains a set of key-value properties that serve to contain the extra information needed to drive the transformation. In this case, the key (*derived attribute*) indicates that we aim at controlling the way a derived attribute has to be mapped, whereas the value (*method*) states the desired option. As a result, this time the *Person_Type* structured type does not contain an *Age* attribute, but a *getAge()* method, that returns an Integer (this was the type of the source property).

The last case is similar but this time the Value of the annotation states that we want to map the derived attribute using a trigger. Therefore, the *Person_Type* structured type contains an *Age* attribute. Additionally, two triggers are created over the corresponding typed table (*Person_Type*). One of them will serve to compute the new value of the *Age* property after the insertion and the other one will do the same after any update.

To support these behaviour we have to code three different ATL rules.

The first one (we can look at it as the default one), *ClassProperty2UDTAttibute*, is shown in Figure 5-35 and maps UML properties to UDT attributes.

```
rule ClassProperty2UDTAttribute {
        from
                prop : UML!Property (
                        not prop.isDerivedAttribute() and
                        not prop.isMultivaluedAttribute() and
                        (
                                prop.type.oclIsTypeOf(UML!DataType) or
                                prop.type.oclIsTypeOf(UML!PrimitiveType)
                        ) and
                        prop.refImmediateComposite().oclIsTypeOf(UML!Class)
                )
        to
                aUDT : SQL2003!Attribute (
                        name <- prop.name,
                        type <- prop.type,
                        structured <- prop.getOwningClass()
                )
}
```

Figure 5-35. ATL Rule ClassProperty2UDTAttibute

The guard uses some helpers to identify the nature of the Property. If it is a multivalued or a derived property, it will be mapped by other rules. As well, it checks its type and whether it belongs to a class (to distinguish from member end associations, that are also properties). If the guard evaluates to true, an SQL:2003 attribute is added in the Structured Type that maps the owning class. The binding to the Structured Type is automatically resolved by ATL engine using the transient links created (see section 4.8.3.2).

The rule for the second options is *DerivedProperty2Method* (Figure 5-36). Its guard ensures that it will match just derived properties that have been annotated to be mapped as methods. Then, a method is added to the corresponding UDT ant the return type is set to be the same of the matched property.

```
rule DerivedProperty2Method {
        from
                prop : UML!Property (
                        (prop.isDerivedAttribute()) and
                        (prop.isMapDerivedAttributeToMethod())
                )
        to
                m : SQL2003!Method (
                        name <- 'get' +
                                        prop.name.substring(1,1).toUpper() +
                                        prop.name.substring(2,prop.name.size()),
                        structured <- prop.getOwningClass(),
                        return_type <- prop.type
                )
}
```

Figure 5-36. ATL Rule DerivedProperty2Method

Finally, the *DerivedProperty2AttributeandTrigger* rule (Figure 5-37) replicates the target pattern of the afore-showed *ClassProperty2UDTAttibute* rule. Likewise, it contains two additional target patterns to create the two triggers to compute the value of the created attribute after insertions and updates. Notice that the *resolveTemp()* ATL operation is used to identify the table over which the triggers have to be created. To that end, it is invoked with two arguments: the first is the containing class of the matched property. The second is the identifier of one of the target patterns that contain the rule that maps such class. Every class is mapped to an UDT plus a Typed Table. Here, we are just interested in such table. So, the *resolveTemp* operation navigates the transient links created during transformation execution to retrieve a reference to such table.

```
rule DerivedProperty2AttributeandTrigger {
        from
                prop : UML!Property (
                        (prop.isDerivedAttribute()) and
                        (prop.isMapDerivedAttributeToTrigger()) and
                        (not (prop->refImmediateComposite().isAbstract))
                )
        to
                a : SQL2003!Attribute (
                        name <- prop.name,
                        type <- prop.type,
                        structured <- prop->refImmediateComposite()
                ),
                tin : SQL2003!Trigger (
                        name <- 'get' + prop.name.substring(1,1).toUpper() +
                                        prop.name.substring(2,prop.name.size()),
                        event <- #INSERT,
                        actionTime <- #AFTER,
                        table <- thisModule.resolveTemp(prop.refImmediateComposite(),'tt'),
                        updateColumns <- a
                ),
                tup : SQL2003!Trigger (
                        name <- 'get' + prop.name.substring(1,1).toUpper() +
                                        prop.name.substring(2,prop.name.size()),
                        event <- #UPDATE,
                        actionTime <- #AFTER,
                        table <- thisModule.resolveTemp(prop.refImmediateComposite(),'tt'),
                        updateColumns <- a
                )
}
```

Figure 5-37. ATL DerivedProperty2AttributeandTrigger

5.3.1.6 One–to–Many (multiple options)

This time we focus on the generic situation illustrated in Figure 5-38, where one element from the source model correspond to several elements on the target one, but multiple options can be chosen: we may map an (A) object to a pair of objects (1) and (2) or to a pair of (1') and (2') objects.

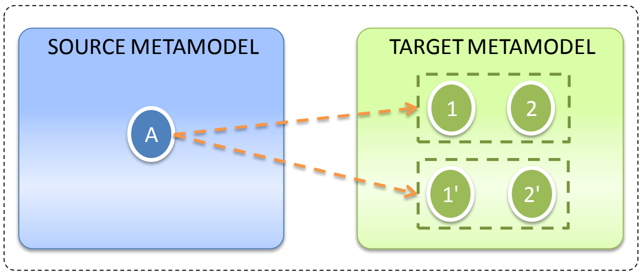
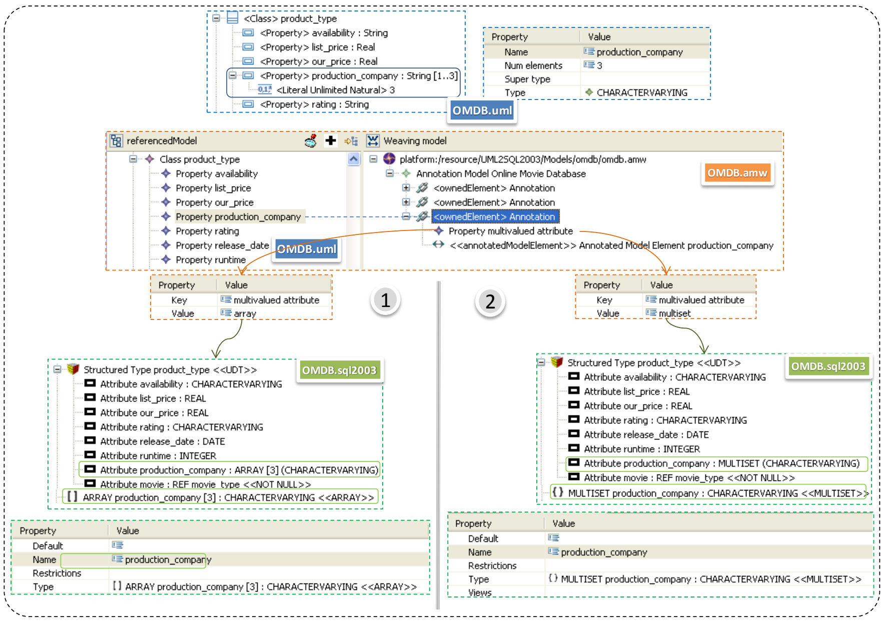


Figure 5-38. Many-to-Many transformation (multiple options)

In the UML2SQL2003 this scenario appears a number of times. For instance, to map multivalued properties to ORDB schemas we have to create both an attribute of a collection type plus the collection type itself. We may choose between two different collection types: MULTISET (dynamically sized) and ARRAY (predefined size). By default, ARRAY types are used, but we can modify this behaviour by annotating the multivalued property. Figure 5-39 shows an example.



```
<Class> product_type
    <Property> availability : String
    <Property> list_price : Real
    <Property> our_price : Real
    <Property> production_company : String [1..3]
        <Literal Unlimited Natural> 3
    <Property> rating : String
```

Property	Value
Name	production_company
Num elements	3
Super type	
Type	CHARACTERVARYING

```
referencedModel
Class product_type
    Property availability
    Property list_price
    Property our_price
    Property production_company
    Property rating
    Property release_date
    Property runtime
```

```
Weaving model
platform:/resource/UML2SQL2003/Models/omdb/omdb.amw
    Annotation Model Online Movie Database
        <ownedElement> Annotation
        <ownedElement> Annotation
        <ownedElement> Annotation
            Property multivalued attribute
            <<annotatedModelElement>> Annotated Model Element production_company
```

Property	Value
Key	multivalued attribute
Value	array

Property	Value
Key	multivalued attribute
Value	multiset

```
Structured Type product_type <<UDT>>
    Attribute availability : CHARACTERVARYING
    Attribute list_price : REAL
    Attribute our_price : REAL
    Attribute rating : CHARACTERVARYING
    Attribute release_date : DATE
    Attribute runtime : INTEGER
    Attribute production_company : ARRAY [3] (CHARACTERVARYING)
    Attribute movie : REF movie_type <<NOT NULL>>
[ ] ARRAY production_company [3] : CHARACTERVARYING <<ARRAY>>
```

Property	Value
Default	
Name	production_company
Restrictions	
Type	[ ] ARRAY production_company [3] : CHARACTERVARYING <<ARRAY>>

```
Structured Type product_type <<UDT>>
    Attribute availability : CHARACTERVARYING
    Attribute list_price : REAL
    Attribute our_price : REAL
    Attribute rating : CHARACTERVARYING
    Attribute release_date : DATE
    Attribute runtime : INTEGER
    Attribute production_company : MULTISET (CHARACTERVARYING)
    Attribute movie : REF movie_type <<NOT NULL>>
{ } MULTISET production_company : CHARACTERVARYING <<MULTISET>>
```

Property	Value
Default	
Name	production_company
Restrictions	
Type	{ } MULTISET production_company : CHARACTERVARYING <<MULTISET>>
Views	

Figure 5-39. Different ways of mapping multivalued attributes

The upper bound multiplicity of the *production_company* Property is 3. Thus, it is multivalued property that can be mapped in two different ways. To choose the one desired for each execution of the transformation, we annotate the Property. This time, the key for the annotation object is *multivalued attribute*. If we set the value to *array* (1), the transformation adds to the target model an ARRAY object. Its type will be the one that maps the type of the source Property. The ARRAY is used to define the type of the OR attribute that maps the UML Property. On the other hand, if we set the annotation value to *multiset* (2), this time the collection type used is a MULTISET. Remember that, in absence of annotation, the default option is to use an ARRAY.

These two different ways of mapping multivalued attributes are encoded in two similar ATL rules shown in Figure 5-40 and Figure 5-41.

```
rule MultivaluedPropertyWithoutGeneratedType2ARRAYAttribute {
	from
		prop : UML!Property (
			(prop.isMultivaluedAttribute()) and
			(prop.isFixedSizeMultivaluedAttribute()) and
			(not prop.isGeneratedMultivaluedType())
		)
	to
		a : SQL2003!Attribute (
			name <- prop.name,
			type <- array,
			structured <- prop->refImmediateComposite()
		),
		array : SQL2003!ARRAY (
			name <- prop.name,
			type <- prop.type,
			num_elements<-prop.upperValue.value,
			schema <- thisModule.PACKAGE()
		)
}
```

Figure 5-40. ATL Rules MultiValuedPropertyWithoutGeneratedType2ARRAYAttribute

Both rules are very similar, their guard matches multivalued UML Properties for which no collection type has already been generated (otherwise, they will be mapped by other rules). Besides, each one filters just those properties to be mapped using a ARRAY (*isFixedSizeMultiValuedAttribute*) or a MULTISET (*isVarSizeMultiValuedAttribute*). Regarding target patterns, the difference lies in the type of the collection object created: one creates an ARRAY (and set its size to the upper bound multiplicity of the matched Property) while the other one creates a MULTISET.

```
rule MultivaluedPropertyWithoutGeneratedType2MULTISETAttribute {
	from
		prop : UML!Property (
			(prop.isMultivaluedAttribute()) and
			(prop.isVarSizeMultivaluedAttribute()) and
			(not prop.isGeneratedMultivaluedType())
		)
	to
		a : SQL2003!Attribute (
			name <- prop.name,
			type <- multiset,
			structured <- prop->refImmediateComposite()
		),
		multiset : SQL2003!MULTISET (
			name <- prop.name,
			type <- prop.type,
			schema <- thisModule.PACKAGE()
		)
}
```

Figure 5-41. ATL Rules MultiValuedPropertyWithoutGeneratedType2MULTISETAttribute

5.3.1.7 Many-to-One (multiple options)

In this case a source pattern composed of several elements could be mapped to two different target patterns, both containing just one element. The scenario is depicted in Figure 5-42.

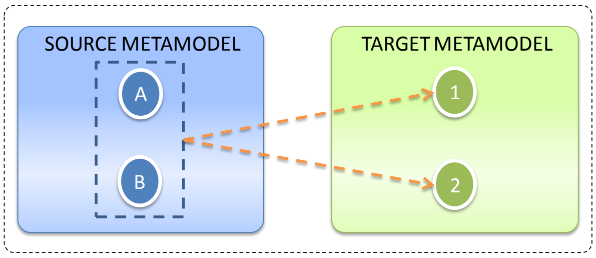
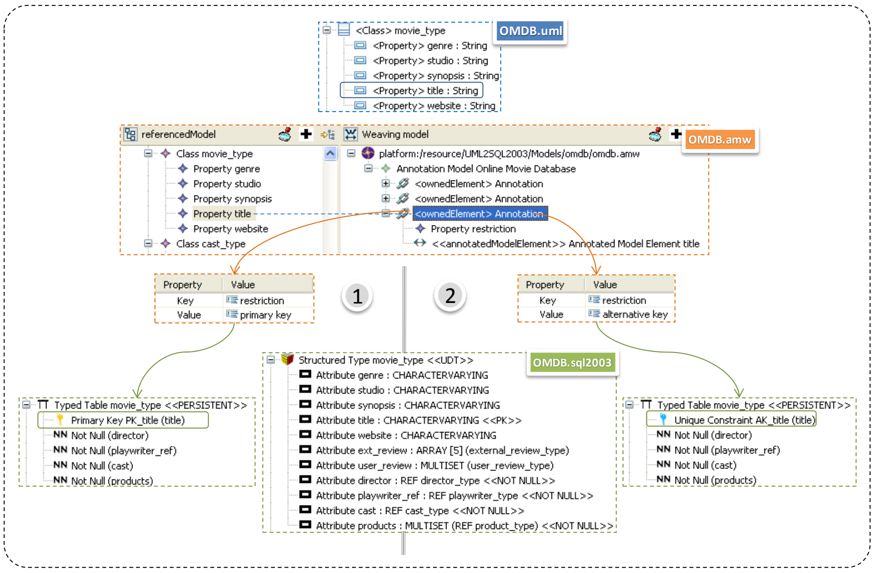


Figure 5-42. Many-to-One transformation (multiple options)

To illustrate this situation we use the mapping of UML properties, that works as identifiers, to ORDB models. Since a pure conceptual model should not specify which are the properties of a Class that should be considered as possible keys, we have to mark the desired Property to be used as unique identifier. To that end, we annotate the property. So, a Class that contains a Property marked as candidate key has to be mapped to a restriction on any Typed Table defined over the UDT that maps the Class. However, the restriction could be a Primary Key or an Unique restriction, depending on the value of the annotation. We illustrate this situation in Figure 5-43.



```
<Class> movie_type
    <Property> genre : String
    <Property> studio : String
    <Property> synopsis : String
    <Property> title : String
    <Property> website : String
```

```
referencedModel
  Class movie_type
      Property genre
      Property studio
      Property synopsis
      Property title
      Property website
  Class cast_type
```

```
Weaving model
  platform:/resource/UML2SQL2003/Models/omdb/omdb.amw
      Annotation Model Online Movie Database
          <ownedElement> Annotation
          <ownedElement> Annotation
          <ownedElement> Annotation
              Property restriction
              <<annotatedModelElement>> Annotated Model Element title
```

Property	Value
Key	restriction
Value	primary key

Property	Value
Key	restriction
Value	alternative key

```
Typed Table movie_type <<PERSISTENT>>
    Primary Key PK_title (title)
    NN Not Null (director)
    NN Not Null (playwriter_ref)
    NN Not Null (cast)
    NN Not Null (products)
```

```
Structured Type movie_type <<UDT>>
    Attribute genre : CHARACTERVARYING
    Attribute studio : CHARACTERVARYING
    Attribute synopsis : CHARACTERVARYING
    Attribute title : CHARACTERVARYING <<PK>>
    Attribute website : CHARACTERVARYING
    Attribute ext_review : ARRAY [5] (external_review_type)
    Attribute user_review : MULTISET (user_review_type)
    Attribute director : REF director_type <<NOT NULL>>
    Attribute playwriter_ref : REF playwriter_type <<NOT NULL>>
    Attribute cast : REF cast_type <<NOT NULL>>
    Attribute products : MULTISET (REF product_type) <<NOT NULL>>
```

```
Typed Table movie_type <<PERSISTENT>>
    Unique Constraint AK_title (title)
    NN Not Null (director)
    NN Not Null (playwriter_ref)
    NN Not Null (cast)
    NN Not Null (products)
```

Figure 5-43. Different ways of mapping unique properties

In this case, the *title* Property of the *movie_type* Class is to be used as unique identifier for *movie_type* objects. Therefore we add an annotation to such Property whose key is *restriction* to indicate so. The type of the restriction to create is set by the value of the annotation: *primary key* or *alternative key*. The former creates a Primary Key over the *movie_type* Typed Table referencing the *title* attribute (1), while the later results in a Unique object with the same bindings (2).

Again, two similar rules serve to address the two options. They are shown in Figure 5-44 and Figure 5-45.

```
rule ClassPropertyPrimaryKey2PrimaryKeyConstraintOnTT {
	from
		a : AMW!Annotation,
		c : UML!Class
		(
			c.generatesTypedTable() and
			a.getReferredProperties()->forAll(prop | c.ownsClassProperty(prop)) and
			a.isPrimaryKeyAnnotation()
		)
	to
		check : SQL2003!PrimaryKey (
			name <- c.getPrimaryKeyName(a.getKeyAttributes()),
			table <- thisModule.resolveTemp(c, 'tt'),
			columns <- a.getReferredProperties()
		)
}
```

Figure 5-44. ATL Rule ClassPropertyPrimaryKey2PrimaryKeyConstraintOnTT

Both of them match any pair of Annotation and Class objects found on the source model if the former annotates a property of the latter. In addition, the Class has to be instantiable class to ensure that it generates a Typed Table in the target model. In that case, the target pattern generates a Primary Key (respectively Unique) object in the target model. Such restriction is binded to the Typed Table that maps the matched Class and refer to all the attributes of such Class that has been annotated to be the Primary Key (respectively the alternative key). Note that this is needed since several properties of a given Class might be annotated in order to obtain a composed Primary (or Alternative) Key.

```
rule ClassPropertyAlternativeKey2UniqueConstraintOnTT {
	from
		a : AMW!Annotation,
		c : UML!Class
		(
			c.generatesTypedTable()
			and a.getReferredProperties()->forAll(prop | c.ownsClassProperty(prop) and
			prop.isAlternativeKeyAttribute()) and a.isAlternativeKeyAnnotation()
		)
	to
		ak : SQL2003!UniqueConstraint (
			name <- c.getAlternativeKeyName(a.getKeyAttributes()),
			table <- thisModule.resolveTemp(c, 'tt'),
			columns <- a.getReferredProperties()
		)
}
```

Figure 5-45. ATL Rule ClassPropertyAlternativeKey2UniqueConstraintOnTT

5.3.1.8 Many-to-Many (multiple options)

The last common scenario we consider is sketched in Figure 5-46. When a pair of A and B objects are found in the source model, they can be mapped to a pair of 1 and 2 objects, or a pair of 1' and 2' objects.

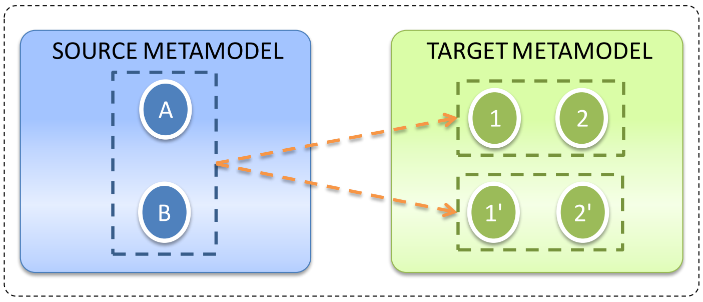


Figure 5-46. Many-to-Many transformation (multiple options)

We have found this scenary a number of times in the transformations developed so far. They specially arise when mapping UML hierarchies to DB models since the later do not support inheritance. Actually, the ORDB model for SQL:2003 does support (partially) such concept, though none commercial product implements such functionality.

To illustrate how we address the development of many-to-many transformations when there are multiple options for the target pattern we will use the example shown in Figure 5-47. Please, note that we have made an extensive study on the different ways of mapping conceptual hierarchies to ORDB models and we have implemented all of them in the transformations bundled in M2DAT-DB (they can be checked in the accompanion CD). Nevertheless, it is not the intention of this dissertation to go deep into the insights of M2DAT-DB. Here we use it just as a reference implementation for M2DAT's specification.

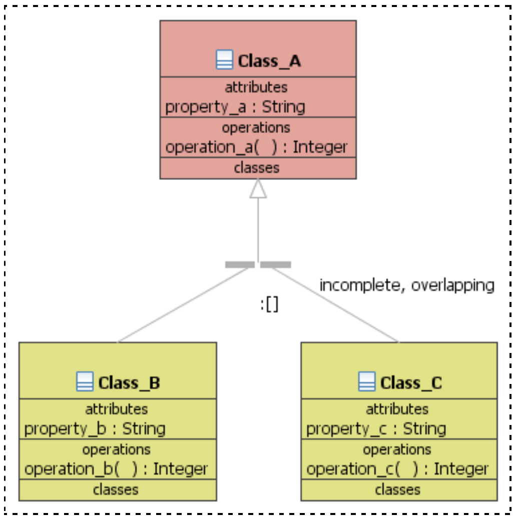
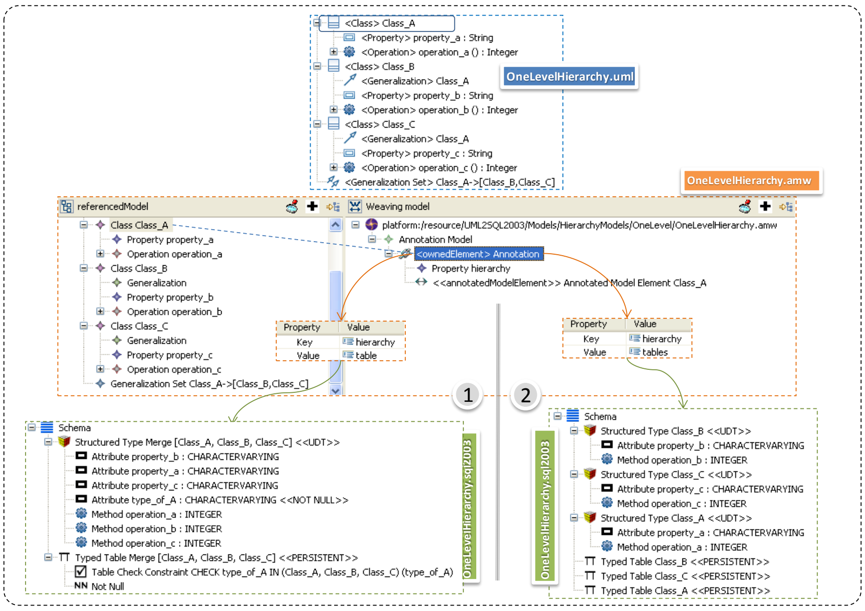


Figure 5-47. Example of UML hierarchy (one level)

In relational data models, the above hierarchy is to be mapped in two different ways: three tables, one per each Class, or one table containing one column per each Property of the three Classes. Figure 5-48 shows the result of encoding these two approaches in the model transformation.

To select the way to map the hierarchy we annotate the parent Class (*Class_A*). This time the key for the annotation is *hierarchy*. If its value is *tables* (1), one merging Structured Type plus one merging Typed Table are created, (so-called *Merge [Class_A*, *Class_B*, *Class_C]*). The Structured Type contains all the attributes and methods of the three classes to map plus a new attribute: *type_of_A*. This is the discriminant attribute that allows identifying the concrete type of each object stored in the merging Typed Table. In addition, a Check constraint is defined over the table to ensure that the discriminant will take an allowed value (Class_A, Class_B or Class_C) and a Not Null constraint to prevent from objects without a concrete type assigned.



```
<Class> Class_A
    <Property> property_a : String
    <Operation> operation_a () : Integer
<Class> Class_B
    <Generalization> Class_A
    <Property> property_b : String
    <Operation> operation_b () : Integer
<Class> Class_C
    <Generalization> Class_A
    <Property> property_c : String
    <Operation> operation_c () : Integer
<Generalization Set> Class_A->[Class_B,Class_C]
```

```
referencedModel
    Class Class_A
        Property property_a
        Operation operation_a
    Class Class_B
        Generalization
        Property property_b
        Operation operation_b
    Class Class_C
        Generalization
        Property property_c
        Operation operation_c
    Generalization Set Class_A->[Class_B,Class_C]
```

```
Weaving model
platform:/resource/UML2SQL2003/Models/HierarchyModels/OneLevel/OneLevelHierarchy.amw
    Annotation Model
        <ownedElement> Annotation
            Property hierarchy
            <<annotatedModelElement>> Annotated Model Element Class_A
```

Property	Value
Key	hierarchy
Value	table

Property	Value
Key	hierarchy
Value	tables

```
Schema
    Structured Type Merge [Class_A, Class_B, Class_C] <<UDT>>
        Attribute property_b : CHARACTERVARYING
        Attribute property_a : CHARACTERVARYING
        Attribute property_c : CHARACTERVARYING
        Attribute type_of_A : CHARACTERVARYING <<NOT NULL>>
        Method operation_a : INTEGER
        Method operation_b : INTEGER
        Method operation_c : INTEGER
    Typed Table Merge [Class_A, Class_B, Class_C] <<PERSISTENT>>
        Table Check Constraint CHECK type_of_A IN (Class_A, Class_B, Class_C) (type_of_A)
        Not Null
```

```
Schema
    Structured Type Class_B <<UDT>>
        Attribute property_b : CHARACTERVARYING
        Method operation_b : INTEGER
    Structured Type Class_C <<UDT>>
        Attribute property_c : CHARACTERVARYING
        Method operation_c : INTEGER
    Structured Type Class_A <<UDT>>
        Attribute property_a : CHARACTERVARYING
        Method operation_a : INTEGER
    Typed Table Class_B <<PERSISTENT>>
    Typed Table Class_C <<PERSISTENT>>
    Typed Table Class_A <<PERSISTENT>>
```

Figure 5-48. Two ways of mapping simple hierarchies from conceptual to ORDB models

On the other hand, if the annotation value is *tables* (2) (default behaviour), three different Structured Types plus three Typed Tables are created. Notice that, in this case, both Class_B and Class_C Structured Types inherits from Class_A Structured Type.

To conclude, Figure 5-49 shows the ATL rule that encodes the first approach, since the later has been already introduced. In fact, each Class is mapped by the *ClassWithoutHierarchy2UDTandTT* rule (see Figure 5-27), giving raise to the three different UDTs plus the three Typed Tables.

```
rule SuperClassWithOneTableHierarchy2UDTandTTandAttributeandCHECKandNOTNULL {
        from
                c : UML!Class (c.isSuperClassWithOneTableHierarchy())
        to
                udt : SQL2003!StructuredType (
                        name <- c.getUDTName(),
                        is_final <- true,
                        is_instantiable <- true,
                        schema <- thisModule.PACKAGE(),
                        super_type <- c.getUDTSuperType(),
                        typed <- tt
                ),
                tt : SQL2003!TypedTable (
                        name <- c.getTypedTableName(),
                        schema <- thisModule.PACKAGE(),
                        structured <- udt,
                        supertable <- c.getSuperTypedTable()
                ),
                a : SQL2003!Attribute (
                        name <- 'type_of_' + c.name,
                        type <- thisModule.ELEMENT_TYPE_STRING(),
                        structured <- c
                ),
                check : SQL2003!TableCheckConstraint (
                        name <- 'Check_Discriminant',
                        expression <- c.getOneTableCheckExpression(),
                        columns <- a,
                        table <- tt
                ),
                notNull : SQL2003!NotNull (
                        table <- tt,
                        columns <- a
                )
}
```

Figure 5-49. ATL Rule SuperClassWithOneTableHierarchy2UDTandTTandAttributeandCHECKandNOTNULL

The guard of the rule invoke the *isSuperClassWithOneTableHierarchy* helper. It restricts the matching to UML Classes acting as parents in a simple hierarchy (i.e. with just one level of descendants) that has been annotated to map the whole hierarchy into just one Structured Type and the corresponding Typed Table.

For each match, the target pattern adds five objects to the target model: the mentioned UDT and Typed Table, the discriminant attribute and the Check and Not Null constraints for the discriminant.

5.3.2 Mapping of Primitive Data Types between PSM Models:

In section 5.2.1.2 we sketched the problems related with modelling the primitive types supported by technological platforms. We provided a solution based on the concept of features. They serve to encapsulate the specific information that has to be provided to specialize a given primitive type for each attribute of such type. Besides, to enhance usability of M2DAT editors, we decided to automatically instantiate all the primitive types in any new model. This way, the user can use them to define the type of the model elements.

Nevertheless, the previous decision entails some challenges for the management of primitive types in model transformations. The improvements on M2DAT editors ensure that new models defined from scratch incorporates all the predefined types. But we need to support the same behaviour for any model obtained as the result of an M2DAT model transformation. In other words, the transformation has to include rules to create all the primitive types of the targeted platform. In this sense, there are two different situations to tackle: PIM2PSM and PSM2PSM transformations.

In the following we show how each one is addressed.

5.3.2.1 Mapping Primitive Types in PIM2PSM transformations

This is the simpler case. We just add a set of matched rules to map each primitive type included in the PIM model, that is, Boolean, String, Integer and Real. For instance, the Figure 5-50 shows the rule to map the *Date* data type. The source pattern matches those *PrimitiveType* objects from the UML model that are *Date* types. The target pattern instantiates the *Datetime* metaclass. The descriptor property is set to DATE to specify the desired concrete type among the family of *Datetime* types. Besides, the new primitive type is nested in the *Schema* object that constitutes the root of the target model. To that end the expression *thisModule.PACKAGE()* resolves the transient link that relates the source Package with the target Schema.

```
rule Date2Date {
        from
                dt : UML!PrimitiveType(dt.isDatePT())
        to
                out : SQL2003!DatetimeType (
                        descriptor <- #DATE,
                        schema <- thisModule.PACKAGE()
                )
}
```

Figure 5-50. ATL Rule Date2Date

Besides, we include an imperative rule to generate the rest of primitive types. In particular, it is an *end point* rule, an ATL rule that is automatically executed just before the transformation execution is finished. Figure 5-51 shows an excerpt of the rule. Note that it only contains a target pattern, i.e. it just adds elements in the target model, without the need for a previous matching with some source pattern. Each element in the target pattern follows the structure of the one from the *Date2Date* matched rule already commented.

```
endpoint rule generateTypes(){
        to
                datetime_timewithtimezone : SQL2003!DatetimeType (
                        descriptor <-#TIMEWITHTIMEZONE,
                        schema <- thisModule.PACKAGE()
                ),
                datetime_timewithouttimezone : SQL2003!DatetimeType (
                        descriptor <-#TIMEWITHOUTTIMEZONE,
                        schema <- thisModule.PACKAGE()
                ),
```

Figure 5-51. ATL Rule generateTypes()

5.3.2.2 Mapping Primitive Types in PIM2PSM transformations

The task of mapping primitive types in PSM2PSM transformations is more challenging. Apart from mapping the primitive types, we need to map the features that each element uses to customize the Primitive Type used (see section 5.2.1.2). We propose two different techniques to tackle these issues: one for mapping the Primitive Type objects and another for the Feature objects. Next, we introduce them using the SQL20032ORDB4ORA transformation bundled in M2DAT-DB (see section 5.1.1). It maps ORDB schemas conforming to the SQL:2003 standard to ORDB schemas for Oracle.

Mapping Primitive type objects

Regarding just primitive types, we can identify the different scenarios summarized in Table 5-2.

Table 5-2. Possible Scenarios for Primitive Types mapping in PSM2PSM transformations

SOURCE MODEL	TARGET MODEL
One Element	One Element
None	One element
Several elements	One Element

The first one is tackled with a matched rule. For instance, the Figure 5-52 shows the ATL rule to map SQL:2003 Character type objects (CharacterStringType.CHARACTER) to Oracle Character type objects (ANSICharacterType.CHARACTER).

```
rule CharacterStringType2Varchar {
        from
                cs : SQL2003!CharacterStringType(cs.descriptor = #CHARACTER)
        to
                ch1 : ORDB4ORA!ANSICharacterType (

                        Descriptor <- #CHARACTER,
                        model <- thisModule.schema
                )
}
```

Figure 5-52. ATL Rrule CharacterStringType2Varchar

We have already shown how the second scenario (none source type to one target type) is solved. An endpoint rule like the one from Figure 5-51 takes care of this issue by instantiating any primitive type considered in the target platform, that is not considered in the source platform.

Finally, the last scenario is the most complex. Here, several source types have to be mapped to the same target type. For instance, both the SQL:2003 NCHAR and CHAR types are mapped to the same Oracle CHARACTER type. In such a situation, we call the source types *mirror* types since they have to return the same target type. This situation is solved with two different steps:

- Mapping one of the mirror types to the desired target type.
- If a source object uses any of the mirror types to define its type, the corresponding target object will use the target type created before.

First step is encoded in a matched rule like the ones already shown in Figure 5-50 and Figure 5-52. As an example, Figure 5-53 summarizes how the second step is carried out to map Parameter objects from SQL:2003 ORDB models to ORBD models for Oracle.

```
rule Parameter2Parameter{
        from
                pIN : SQL2003!MethodParameter
        to
                pOUT : ORDB4ORA!MethodParameter
                (
                        Name <- pIN.name,
                        Type <- if pIN.type.isMirrorType() then
                                        pIN.type.mirrorType()
                                else
                                        pIN.type
                                endif
                )
}
```

Figure 5-53. SQL:2003 to ORDB4ORA --> ATL Rule Parameter2Parameter

We have already explained how target elements are referenced in ATL code. To that purpose, ATL replaces references to a source element by a reference to the corresponding target element. We can not proceed this way in this case since the target type has not just one corresponding source element but several (the mirror types). So, whenever a reference to a primitive type has to be made, we check whether it is a mirror type. If so, instead of using the reference as-is, we invoke the *mirrorType* helper directly. From the set of mirror types that has to be mapped to the same target type, the helper returns the type that is used by the matched rule that creates the corresponding target type.

Mapping Feature objects

Once the primitive types are correctly mapped, we need to address the mapping of the features that each structural component (attribute, field or column) uses to adapt the type to its specific needs (see section 5.2.1.2). Notice that only those source features with a corresponding feature in the target model could be mapped. Figure 5-54 shows part of the solution. In particular, it shows the rule to map SQL:2003 attributes to Oracle attributes.

```
rule Attribute2Attribute {
        from
                attIN : SQL2003!Attribute
        to
                attOUT : ORDB4ORA!Attribute (
                        Name <- attIN.name,
                        Type <- if attIN.type.isHiddenType() then
                                        attIN.type.mirrorType()
                                else
                                        attIN.type
                                endif,
                        structured <- attIN.structured,
                        features <- attIN.features->select(f|f.haveLegalTarget())->collect(f|thisModule.Feature2Feature(f))
                        )
}
```

Figure 5-54. SQL:2003 to ORDB4ORA --> ATL Rule Attribute2Attribute

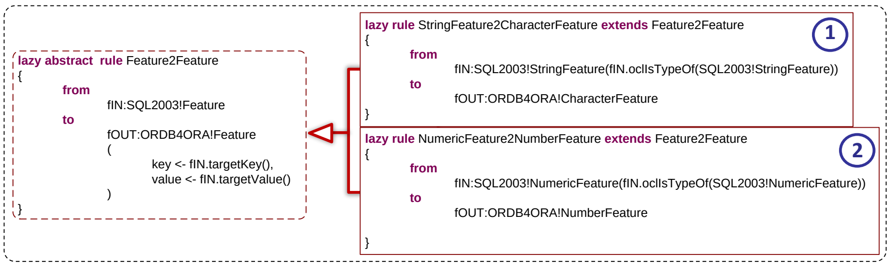
Whenever an Attribute is mapped, its features have to be mapped as well. To that end, we first select just those features that have a correspondent feature on the target metamodel. To filter them we use the *haveLegalTarget()* helper. Then, we invoke the rule that creates the target feature (*Feature2Feature()*).

Indeed, the *Feature2Feature* rule, shown in Figure 5-55, is an abstract rule. It maps the source key-value pair to the target key-value pair. To that end, two different helpers return the target key and the target value for each source key and source value. Taking advantage from ATL rule inheritance, the rule is later specialized for each family of primitive types.

```
lazy abstract  rule Feature2Feature
{
        from
                fIN:SQL2003!Feature
        to fOUT:ORDB4ORA!Feature
                (
                        key <- fIN.targetKey(),
                        value <- fIN.targetValue()
                )
}
```

Figure 5-55. SQL:2003 to ORDB4ORA --> ATL Rule Feature2Feature

For instance, Figure 5-56 shows how the rule is specialized for the families of String (1) and Numeric (2) primitive types.



```
lazy abstract  rule Feature2Feature
{
        from
                fIN:SQL2003!Feature
        to
                fOUT:ORDB4ORA!Feature
                (
                        key <- fIN.targetKey(),
                        value <- fIN.targetValue()
                )
}
```

```
lazy rule StringFeature2CharacterFeature extends Feature2Feature
{
        from
                fIN:SQL2003!StringFeature(fIN.oclIsTypeOf(SQL2003!StringFeature))
        to
                fOUT:ORDB4ORA!CharacterFeature
}
```

```
lazy rule NumericFeature2NumberFeature extends Feature2Feature
{
        from
                fIN:SQL2003!NumericFeature(fIN.oclIsTypeOf(SQL2003!NumericFeature))
        to
                fOUT:ORDB4ORA!NumberFeature

}
```

Figure 5-56. SQL:2003 to ORDB4ORA --> Instantiating Feature2Feature ATL rule

5.3.3 Documenting ATL Transformations

So far, we have already presented how model transformations are addressed when developing M2DAT's modules. In this section we would like to present another minor improvement introduced in M2DAT regarding the development of ATL model transformations.

One of the main drawbacks of current model transformation languages is available documentation. Since they are still too recent, the most of the effort is dedicated to build and improve the transformation engine while almost no effort is dedicated to document it, a crucial factor regarding final adoption of the language. Although ATL is the best of existing languages in this sense, we have added an improvement on M2DAT regarding documentation of ATL transformations. We firmly believe it contributes to improve M2DAT usability.

Constant addition of comments in the code is a good practice. However, when the transformation gets too large or complex, documenting could turn out to be a tedious task. One possible improvement is the use of automatic documentation mechanisms, like Javadoc [394], the most recognised and adopted way of documenting source code. Following this approach, we have built a utility similar to Javadoc to generate HTML doc from ATL source code, so-called

ATLDoc. It is based on two main points: the comment format and the HMTL output style.

- **Comment Format**: we will use the modular nature of ATL to define enriched comments for each ATL block (rules, helpers, etc.). To associate a meaning to each comment we lean on a little grammar encoded in an XML file. It serves to identify the beginning and finish of each comment, plus the different subsections that it owns. Figure 5-57 shows an example of such file.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
 <ATLDOC>
       <Comment>
          <Begin>--BEGIN DOC
          </Begin>
          <End>--END DOC</End>
          <Sections escapeChars="#">
                <Title escapeChars="" name="PRECONDITION">
                  </Title>
                  <Title escapeChars="" name="About">
                        <Subtitle>@name</Subtitle>
                        <Subtitle>@version</Subtitle>
                        <Subtitle>@domains</Subtitle>
                        <Subtitle>@authors</Subtitle>
                        <Subtitle>@date</Subtitle>
                        <Subtitle>@description</Subtitle>
                  </Title>
                  <Title escapeChars="" name="DESCRIPTION">
                        <Subtitle>@CONTEXT</Subtitle>
                        <Subtitle>@INPUTS</Subtitle>
                        <Subtitle>@RETURN</Subtitle>
                        <Subtitle>@LIBRARIES</Subtitle>
                        <Subtitle>@AUTOR</Subtitle>
                  </Title>
          </Sections>
     </Comment>
     <Code>
      <startWith>helper</startWith>
      <startWith>rule</startWith>
      <startWith>lazy</startWith>
      <startWith>entrypoint</startWith>
      <startWith>endpoint</startWith>
      <startWith>uses</startWith>
      <startWith>library</startWith>
      <startWith>module</startWith>
      <startWith>abstract</startWith>
      <startWith>unique</startWith>
     </Code>
</ATLDOC>
```

Figure 5-57. ATLDoc Template

This way, the <Begin> tag identifies the beginning of a comment while the <End> tag identifies its finishing. The *escapeChars* attribute of the <Sections> tag denotes each subsection inside a comment. In turn, each Section has a title denoted by the *name* attribute of the <Title> tag and can have subsections, marked by the <Subtitle> tag.

For instance, the ATL file from Figure 5-58 has been coded according to the default template.

```
-- @atlcompiler atl2006
-- @nsURI UML=http://www.eclipse.org/uml2/2.1.0/UML

--BEGIN DOC
--#About
-- @name         UML_constants
-- @version      1.0
-- @domains      database, dsl, sql2003, uml, mda, transformation,
--               metamodel, model
-- @authors      Alejandro Galindo (Universidad Rey Juan Carlos)
-- @date         24-03-2008
-- @description  Esta libreria ATL contiene las constantes utilizadas en las
--               transformaciones realizadas desde un modelo conforme
--               al metamodelo UML.
--END DOC

library UML_constants;

-- Dentro de las librerias de ATL no se permite definir atributos o constantes.
-- Entonces, las constantes hay que definirlas como helpers.

--BEGIN DOC
--#DESCRIPTION
--Constante asociada al tipo primitivo de UML para representar cadenas de texto.
--END DOC
helper def : TYPE_STRING() : String = 'string';

--BEGIN DOC
--#DESCRIPTION
--Constante asociada al tipo primitivo de UML para representar caracteres.
--END DOC
helper def : TYPE_CHAR() : String = 'char';
```

Figure 5-58. Excerpt from UML_Constants.ATL file

- **Output Style**: the style of the documentation file is encoded in a CSS style sheet. As Figure 5-59 shows, when ATLDoc is invoked, the ATL file containing the structured comments is processed by the ATLDoc utility according to the active template. The output will be an HTML file containing all the source code with the comments section interleaved in a friendly interface according to the styles defined in the CSS file.

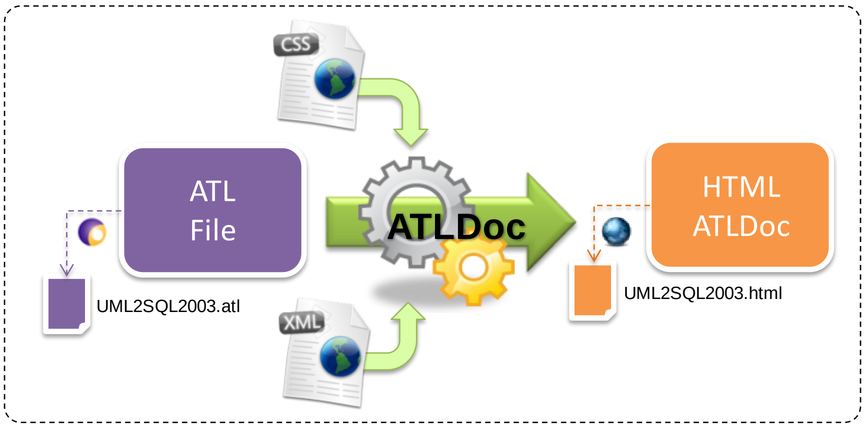
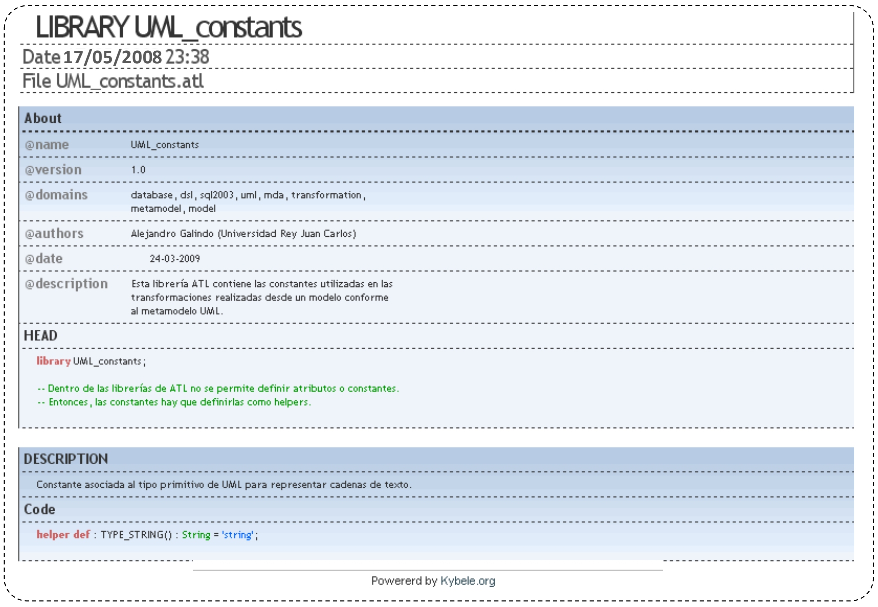


Figure 5-59. ATLDoc overview

This way, after invoking ATLDoc over the ATL file shown in Figure 5-58, an HTML preserving the name with a different suffix is created in the same folder. An excerpt of the file is shown in Figure 5-60.



LIBRARY UML_constants

Date 17/05/2008 23:38

File UML_constants.atl

About

@name	UML_constants
@version	1.0
@domains	database, dsl, sql2003, uml, mda, transformation, metamodel, model
@authors	Alejandro Galindo (Universidad Rey Juan Carlos)
@date	24-03-2009
@description	Esta librería ATL contiene las constantes utilizadas en las transformaciones realizadas desde un modelo conforme al metamodelo UML.

HEAD

```
library UML_constants;

-- Dentro de las librerías de ATL no se permite definir atributos o constantes.
-- Entonces, las constantes hay que definirlas como helpers.
```

DESCRIPTION

Constante asociada al tipo primitivo de UML para representar cadenas de texto.

Code

```
helper def : TYPE_STRING() : String = 'string';
```

Powererd by Kybele.org

Figure 5-60. ATLDoc generated file: UML_Constants.html

In essence, it is the same ATL file with extra features. Next to the name of the module (library in this case), the date of creation plus the name of the ATL file are added on the header. Besides, the name of each section and sub-section of each comment are bolded, while reserved words and primitive type values (like strings) are displayed in a different colour.

To conclude, it is worth mentioning that in the next future we can extend ATLDoc to generate code documentation in other formats, such as PDF, CHM or RTF.

5.3.4 On the Development of Model Transformations

The previous sections serve to prove that using ATL and AMW we are able to address any given scenario that may arise in the context of model transformations. This section is just an attempt to capture some findings, thoughts and lessons learned while developing the model transformations bundled in M2DAT's reference implementation (M2DAT-DB). As well, we would like to provide with some comments on the comparison between ATL and existing implementations of the QVT standard.

5.3.4.1 Some generic reflections

Regarding the language used to code the transformations, the hybrid approach of ATL has turned out to be the most suitable. On the one hand, adopting a declarative style gets rid of part of the complexity inherent to the development of model transformations. Working this way, when you are coding the rules that map a particular metaclass you do not have to wonder about the rest of the metaclasses in the metamodel.

Besides you do not have to worry about how target elements are created, you just need to specify the relationships that must hold between source and target model. The rest is undertaken by the transformation engine.

As well, this eases the task of traceability management. Indeed, the transformation engine uses transient links to establish the bindings needed between target elements. Those transient links stores the information on which target elements have been created to map each source element. Therefore, you just need to persist those transient links if you want your traceability information to be registered.

Likewise, there is no need to care about the order in which rule is executed since declarative programming has no explicit order. The transformation enforces that all the relationships between source and target elements encoded in the rules will hold after execution. But nothing has to be sais about the order in which it has to be done.

Finally, matching of source elements is automatically done by the engine. With imperative programming you would need to code huge loops to navigate the whole source model in order to find all the elements of a given type, and then check if they conform to whatever condition you may impose to map them. In declarative programming, this is done by free by the transformation engine. You just need to specify which condition must be checked over each type of element found on the source model.

Nevertheless, when you are coding complex model transformations you will need for sure some aid from imperative constructions. As we have shown so far, it is very common the situation in which you need to create some "new" elements in the target model. That is, elements for which no relation to a source element must hold. For instance, when moving from PIM to PSM, you need to create the built-in types of the given platform in your model in order to define the type of the target elements. However, those types do not have a correspondence with anything from the source model. In this case, you need to explicitly create the types. To that end, you need an imperative construction.

In this sense, it might be remarkable the fact that even the standard, QVT, proposes tow different languages to support both programming paradigms, allowing to use some imperative operations on your declarative rules.

From our experiences, we can state that the main problem of adopting a declarative approach was changing our mindset. Moving from the imperative programming style to the declarative one resulted quite challenging in the beginning. You tend to twist the declarative rules to make them look like imperative. However, once you have acquired some skills with declarative programming, you immediately come to the conclusion that it is the most suitable for model transformation development. We might say that you start to "think on declarative".

Another issue related with declarative approaches is performance. We have confirmed that, in presence of large models, the performance is rather slow. However, we should take into account the novelty of model transformation engines. They are constantly improved. Thus, we have to let them some time to check if those improvements solve this drawback.

Besides, we have mentioned that traceability management is easier to address in declarative transformations. However, we are using an hybrid approach because we need some imperative constructions. Obviously, this hampers the management of traceability links. In fact, with "new" objects no traceability link could be created, since they do not have relation with any element from the source model. Thus, new techniques are needed to solve this problem. Meanwhile, some temporal fixes can be used. For instance, as we have done so far with built-in types, associate them with the root element of the source model. This way, since every EMF model has to include a root element, we can bind the "new" elements in the target model with the root element from the source model. This way, we can create traceability links also for "new" elements.

Finally, we would like to mention that, our experiences so far has confirmed that a visual notation for developing model transformations is, at best, not enough to develop complex transformations. Think on graph-based transformations, a formal way of defining visual transformations. Though they are mainly visual languages, we have realized that in the most cases, they need to include textual add-ins in their graph-rules to be able to support the whole transformation (ATOM^3 works this way, and even VIATRA itself is still developing support for visual rules). In the end, some expressions are just impossible to model in a visual way. Or if possible, the effort needed is not worthwhile when compared with that needed to specify the same expression in a textual language.

Regarding annotation models, we are pretty sure about the need of having mechanisms to mark source models in order to support design decisions. A complete automatic process from CIM to working-code is nor feasible, neither acceptable. Indeed, this was recognised in the early versions of MDA guide and all along MDE literature. However, as we have argued along this dissertation, marking the model itself means polluting it with concepts from other domains. We stated that the best option was to use annotation models and we have proved that weaving models serve in an efficient and usable way as annotation models. The screen captures has shown that AMW integrates perfectly with EMF tree-like editors and provides with an intuitive and easy to use, yet powerful annotating mechanism. Besides, the good coupling of AMW and ATL eases processing the annotation in the model transformations.

5.3.4.2 ATL vs QVT implementations

As we have already mentioned, we wanted to ensure that existing implementations of QVT standard were not enough to develop complex transformations. Nevertheless, QVT is not a language but a family of languages. Therefore, in order to test how the different programming styles fit with the task of developing model transformations, we use two of the languages from the QVT

family: QVT-Operational Mappings (imperative style) and QVT-Relations (declarative style). Thus, we replicated an ATL transformation with two of the most mature existing implementations of QVT: mediniQVT and QVTo from OpenCanarias. This section presents some highlights gathered from such tests.

Please, note that the objective of this dissertation is not to carry out an exhaustive comparative between model transformation languages. We just want to be able to select the best model transformation language for our purposes. In this sense, the level of complexity is quite relevant. Besides, the ability to make these kind tests in the framework of M2DAT serves to prove the utility of M2DAT as an integrated framework where emerging technologies may be tested and evaluated. Again, this is due to the open nature of M2DAT and the underlying EMF framework.

All this given, our main concern with QVT implementations regards usability. The use of the evaluated languages hampers the code of the transformation. This fact is mainly due to the difference ways of using **tracing information**.

The ATL engine stores the tracing information between every source element and the corresponding target elements in transient links. This information is used by the ATL-VM [180]. Each time one transformation rule is matched, a new tracing link is created between the matched source element and all its corresponding target elements. Subsequently, when a transformation rule implicitly requires the target elements produced for a different rule, the ATL-VM automatically resolves the dependency using the tracing links. We use a code excerpt (Figure 5-61) from one of the transformations carried out in this thesis to show how this works.

```
rule Class2UDT {
 from
   c : UML!Class
 to
   udt : ORDB4ORA!StructuredType(
         name <- c.getUDTName(),
         typed <- tt
         ………….
  ),
 tt : modeloOR!TypedTable(
         name <- c.getTypedTableName()
}

rule Property2Attribute {
   from
     p:UML!Property (not p.isDerived and not p.isMultivalued() and p.refImmediateComposite().oclIsTypeOf(UML!Class))
   to
     a : ORDB4ORA!Attribute(
         name <- p.name,
         type <- p.type,
         structured <- p->refImmediateComposite())
}
```

Figure 5-61. ATL Code Excerpt: Class2UDT and Property2Attribute mapping rules

Rule *Class2UDT* (lines 1-9) maps every *class* from the source model to an *StructuredType* plus a *Typed Table* on the target model. Rule *Property2Attribute* (14-22) maps every *Property* of a *UML!Class* to an *Attribute* of the corresponding *StructuredType* in the target model. The binding *structured <- p.refImmediateComposite()* returns a reference to such *StructuredType*. In fact, the binding returns a reference to the owning class of the property. The ATL-VM replaces it for a reference to the corresponding *StructuredType*. To do so, the ATL-VM uses the internal traceability links handled during the execution of the transformation. This way, the reference to the owning class is replaced by a reference to the *StructuredType* that generates the *Class2UDT* rule that maps the class.

The absence of those transient links in QVT studied engines hampers the development of model transformations. As noticed before this fact forces somehow to follow a not purely declarative style. When you are developing a model transformation in the declarative style, you should be able to code each rule without worrying about how the rest of elements from the source model are mapped. In the previous example, you do not have to worry about *Properties* mapping when coding the *Class2UDT* rule. In contrast, we can have a look at the equivalent mapping rules for mediniQVT (Figure 5-62).

```
1   top relation Class2UDT {
2
3     n : String;
4
5     checkonly domain uml c : uml::Class {name = n};
6
7     enforce domain ordb4ora s : ORDB4ORA::StructuredType
8      {name = n + ' <<udt>>', typed = t : ORDB4ORA::TypedTable {name = c.name}, model = getModel()};
9
10    when {PackageToModel(getPackage(), getModel());(c.generalization->first()->oclIsUndefined())=true;}
11
12    where {PropertyToAttribute2 (c, s); PropertyDerivedToMethod (c, s);
13    }
14
15  }
16
17  relation PropertyToAttribute2 {
18
19    an : String;
20    pn : String;
21
22    checkonly domain uml c : uml::Class
23      {ownedAttribute = p : uml::Property {name = an, type = upt : uml::PrimitiveType {name = pn}}
24    };
25
26    enforce domain ordb4ora s : ORDB4ORA::StructuredType
27      {attribute = a : ORDB4ORA::Attribute {name = an, type = opt : ORDB4ORA::PrimitiveType {model = getModel()}}
28    };
```

Figure 5-62. mediniQVT Code Excerpt: Class2UDT and Property2Attribute mapping rules

Notice that when you are coding the mapping rule for Classes (lines 1-13), you have to keep in mind Class' properties since the *Properties2Attributes* mapping rule has to be invoked from the *Class2UDT* rule (line 12).

The same happens to any other model element nested in a Class, like methods or association ends. Clearly, this way of programming model transformations is not recommended, at least for us, and it differs from the purely declarative style.

Another limitation of existing QVT implementations (at least, of QVT-Operational Mappings) lies in the way they handle the target model. Any element added to the target model has to be initially nested on the root element and later reallocated using a property of any element from that same model.

A global remark that can be made about QVT-Relational and QVT-Operational Mapping is related with the approach adopted by each one. While QVT-Relations adopts a purely declarative approach, QVT-Operational Mappings adopt an imperative one. As we have argued in section 4.4.2.3, we believe that none of them are the best way to address the development of model transformations. While using just declarative constructions is not feasible in some scenarios, sticking to imperative constructions results in too much verbosity and very complex transformations.

Finally, as MOF QVT Revision Task Force shows (http://www.omg.org/issues/qvt-rtf.html), QVT specification presents serious drawbacks and is rather susceptible of being revisited as long as QVT implementers advance in their work and more inconsistencies are detected. Hence, a complete, efficient and reliable QVT implementation is still to come.

5.4 Code Generation in M2DAT

Previous section has focused on the development of model-to-model transformations in M2DAT. Indeed code generation is also a model transformation, but this is a model-to-text transformation.

However, as we have mentioned several times along this dissertation, we believe that model-to-model transformation is the cornerstone of model-driven development. Indeed, model-to-text transformation is just about serializing models. If you start from a well-defined and precise PSM, though relevant, it is a less challenging task.

In contrast, model-to-model transformations have to deal with changes in abstraction levels and/or domains, what adds a lot of complexity to the task, if you want your model transformations to be complete. That is, you need to be very careful when developing transformation to capture all the information from the source model and to translate it properly to the set of abstraction supported by the target metamodel.

All this given, this section will provide just with some insights on how model-to-text transformations have been developed in M2DAT so far. To that end we use the MOFScript program for generating SQL scripts conforming to SQL:2003 standard from SQL:2003 ORDB schemas.

5.4.1 Using MOFScript for code generation purposes

In front of the declarative approach of ATL (and the vast majority of existing model to model proposals), model to text transformation engines take the form of imperative programming languages. In fact, a MOFScript script is a parser for models conforming to a given metamodel. While it parses the model structure, it generates a text model based on transformation rules. On a second phase this text model is serialized into the desired code. This way, the script uses the metamodel to drive the navigation through the source model, just as an XML Schema drives the validation of an XML file. As a matter of fact, every model is persisted in XMI format, an XML syntax for representing UML-like (or MOF) models.

The program that implements the model to text transformation is basically a model parser. It navigates the structure of the model, generating a formatted output stream. In this case, the model os the ORDB model while the output stream is the SQL script that implements the modeled DB schema. In the following we introduce this script showing some code excerpts. The reader is referred to [262] for more information on how to configure MOFScript execution.

As showed below, a *main* function is the entry point for the script. It includes a set of rules for processing each possible type of element that can be found in the source model (so-called context types in MOFScript). Besides, we include the *eco* parameter in the script header to specify which the input metamodel is. To that end we use the URI that identifies the metamodel we have presented in section 5.2.1.1.

```
texttransformation codigo (in eco:"http://SQL2003.ecore") {

  eco.Schema::main(){
   var nombre:String

   if (self.name.size()=0)
         nombre="codigo_SQL2003.sql"
   else
         nombre=self.name + "_SQL2003.sql"
   file (nombre)

   println("CREATE SCHEMA " + self.name + ";")
   println("")


   //code generation for Structured Types
   self.datatypes->forEach(s:eco.StructuredType)
   {
                  s.generateStructured()
                  println("")
   }
```

Figure 5-63. MOFScript code excerpt: heading

Next, a transformation rule is defined for each context type. For simple rules, we code the rule inside the *main* body whilst the complex ones are coded by means of auxiliary rules. Those functions are invoked from the *main* body.

For instance, the rule for Structured Types creation is probably the most complex one since it encapsulates a lot of semantics. Thus, it is coded in the *generateStructured* auxiliary function. The *main* body invokes it for every Structure Type object found in the source model.

The code excerpt shown in Figure 5-64 presents the beginning of the *generateStructured* rule.

```
eco.StructuredType::generateStructured() {
   var texto:String=""
   var mCount:integer= self.method.size()
   var currentMethod:integer=0
   var i:integer=0

   if (self.super_type.name.size()=0)
        texto="CREATE " + self.name + " AS"
   else
        texto="CREATE  " + self.name +
                 " UNDER " + self.super_type.name+ " AS"
```

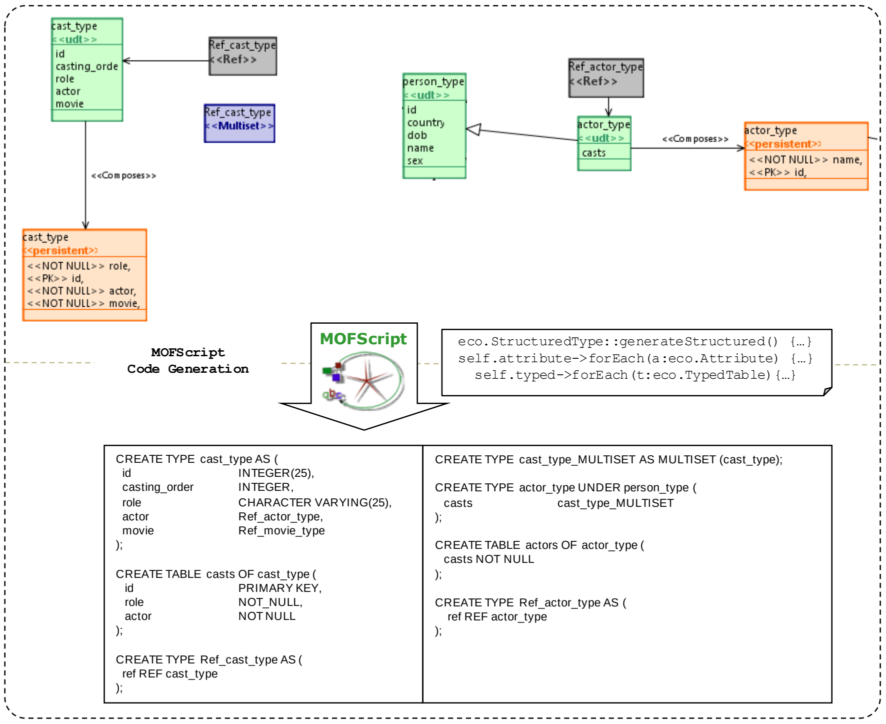
```
      if(self.attributes.size() == 0 and self.method.size() == 0)
               print(texto + "()")
      else
      {
               println(texto + "\n(")

   //adds UDT's attributes
      self.attributes ->forEach(a:eco.Attribute) {
        i=i+1
       a.generateAttribute()
       if(i==self.attributes.size())
        println("")
        else
        println(",")
      }
```

Figure 5-64. MOFScript code excerpt: GenerateStructuredType rule

First, the auxiliary variable that will store the SQL code is initialized. Next, we add the SQL code to start the creation of the structured type, distinguishing those types that inherit from any other type from those that do not. Then, the script checks whether the Structured Type contains any attribute. If so, it navigates the collection of attributes invoking the corresponding rule (*generateAttribute*) and so on.

To conclude this section, Figure 5-65 shows a piece of the SQL code generated for the case study.



```
eco.StructuredType::generateStructured() {…}
self.attribute->forEach(a:eco.Attribute) {…}
  self.typed->forEach(t:eco.TypedTable) {…}
```

```
CREATE TYPE  cast_type AS (
  id                    INTEGER(25),
  casting_order         INTEGER,
  role                  CHARACTER VARYING(25),
  actor                 Ref_actor_type,
  movie                 Ref_movie_type
);

CREATE TABLE  casts OF cast_type (
   id                   PRIMARY KEY,
   role                 NOT_NULL,
   actor                NOT NULL
);

CREATE TYPE  Ref_cast_type AS (
  ref REF cast_type
);
```

```
CREATE TYPE  cast_type_MULTISET AS MULTISET (cast_type);

CREATE TYPE  actor_type UNDER person_type (
  casts                 cast_type_MULTISET
);

CREATE TABLE  actors OF  actor_type (
  casts NOT NULL
);

CREATE TYPE  Ref_actor_type AS (
   ref REF actor_type
);
```

Figure 5-65. SQL Generated Code exceprt

The upper side is a screen capture of the developed graphical editor. It shows an extract from the OR model of the case study. Specifically the *cast* and *actor* types and the corresponding typed tables, next to the REF types created from them as well as the collection types. This code is generated by the execution of the mapping rules listed in the annotation beside.

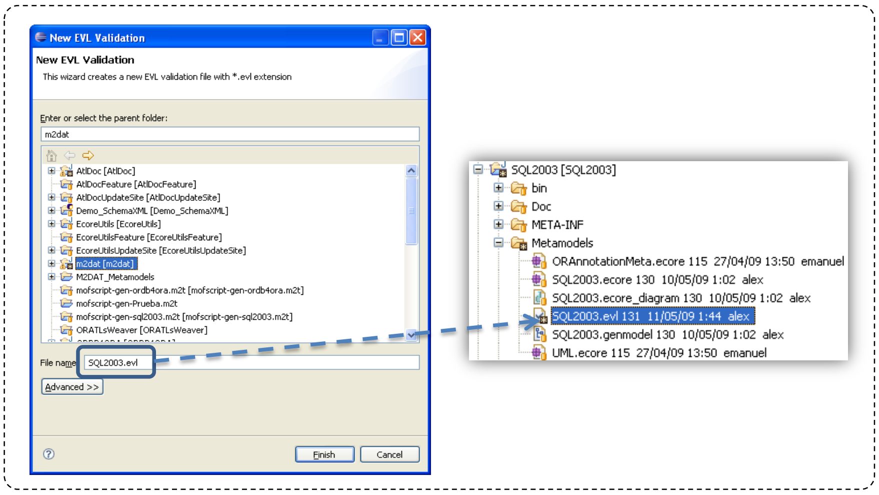
5.5 Validating models in M2DAT

Section 4.7 discussed the election of EVL as the way to integrate OCL-based model validation in the DSLs bundled in M2DAT. With EVL, the constraints to be checked are defined at metamodel level. The Epsilon engine provides with the mechanisms to add the evaluation of such constraints over any terminal model conforming to such metamodel.

We have opted for batch validation instead of live validation since live validation presents the recurrent problem of modelling objects going out of a valid state only to be eventually placed back into a valid state.

In the following we show how a constraint is defined to ensure that any Schema object will have a name and the rules to define such name. In addition, we show the result of validating an .sql2003 file when such constraint is not satisfied.

First thing to do is to code the corresponding invariant. into an .evl file. After installing Epsilon, wizards to create EVL files are available (see Figure 5-66).



New EVL Validation

This wizard creates a new EVL validation file with *.evl extension

Enter or select the parent folder: m2dat

```
AtlDoc [AtlDoc]
AtlDocFeature [AtlDocFeature]
AtlDocUpdateSite [AtlDocUpdateSite]
Demo_SchemaXML [Demo_SchemaXML]
EcoreUtils [EcoreUtils]
EcoreUtilsFeature [EcoreUtilsFeature]
EcoreUtilsUpdateSite [EcoreUtilsUpdateSite]
m2dat [m2dat]
M2DAT_Metamodels
mofscript-gen-ordb4ora.m2t [mofscript-gen-ordb4ora.m2t]
mofscript-gen-Prueba.m2t
mofscript-gen-sql2003.m2t [mofscript-gen-sql2003.m2t]
ORATLsWeaver [ORATLsWeaver]
```

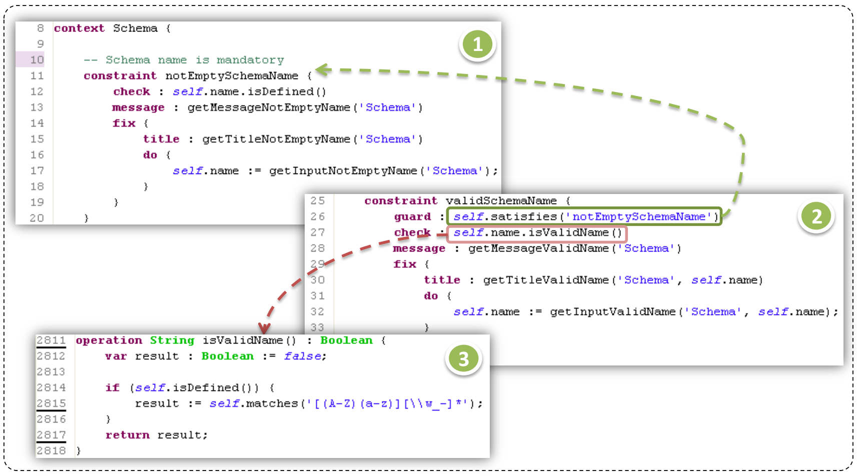
File name: SQL2003.evl

```
SQL2003 [SQL2003]
  bin
  Doc
  META-INF
  Metamodels
    ORAnnotationMeta.ecore 115  27/04/09 13:50  emanuel
    SQL2003.ecore 130  10/05/09 1:02  alex
    SQL2003.ecore_diagram 130  10/05/09 1:02  alex
    SQL2003.evl 131  11/05/09 1:44  alex
    SQL2003.genmodel 130  10/05/09 1:02  alex
    UML.ecore 115  27/04/09 13:50  emanuel
```

Figure 5-66. Creating EVL files

Figure 5-67 shows the EVL code that implements the invariant to control that every Schema object owns a valid name.

The *notEmptySchemaName* invariant prevents from void Schema names (1). Next, the *validSchemaName* is evaluated over those schema objects for which the previous invariant evaluates to true (2). It checks if the name fulfil the specified construction rules. Those rules are summarized in a regular expression encoded in the *isValidName()* operation (3). If the name is not correct, then the fix code is executed. It shows a *getTitleValidName* message inviting the user to correct the error detected. If the user does show, the *getInputValidName* operation will show an input box where the user can enter a new name for the Schema object.



```
context Schema {

    -- Schema name is mandatory
    constraint notEmptySchemaName {
        check : self.name.isDefined()
        message : getMessageNotEmptyName('Schema')
        fix {
            title : getTitleNotEmptyName('Schema')
            do {
                self.name := getInputNotEmptyName('Schema');
            }
        }
    }
```

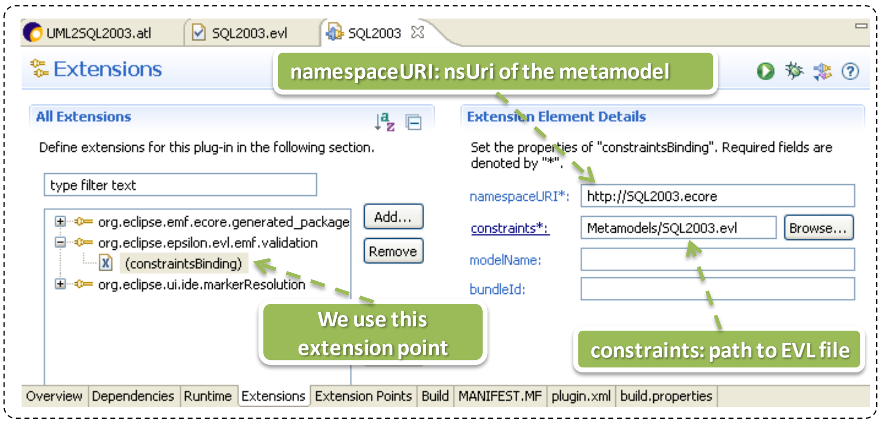
```
    constraint validSchemaName {
        guard : self.satisfies('notEmptySchemaName')
        check : self.name.isValidName()
        message : getMessageValidName('Schema')
        fix {
            title : getTitleValidName('Schema', self.name)
            do {
                self.name := getInputValidName('Schema', self.name);
            }
```

```
operation String isValidName() : Boolean {
    var result : Boolean := false;

    if (self.isDefined()) {
        result := self.matches('[(A-Z)(a-z)][\\w_-]*');
    }
    return result;
}
```

Figure 5-67. EVL Invariant to enforce Schema names consistency

Then, the .evl file has to be associated to the plug-in that contains the code for handling models. To that end, the configuration file of the plug-in that implements the editor is used. We have to specify that the editor extends the validation plug-in from Epsilon as well as the URI assigded to the metamodel of the DSL plus the path to the .evl file containing the constraints definition (Figure 5-68).



Define extensions for this plug-in in the following section.

org.eclipse.emf.ecore.generated_package
org.eclipse.epsilon.evl.emf.validation
(constraintsBinding)
org.eclipse.ui.ide.markerResolution

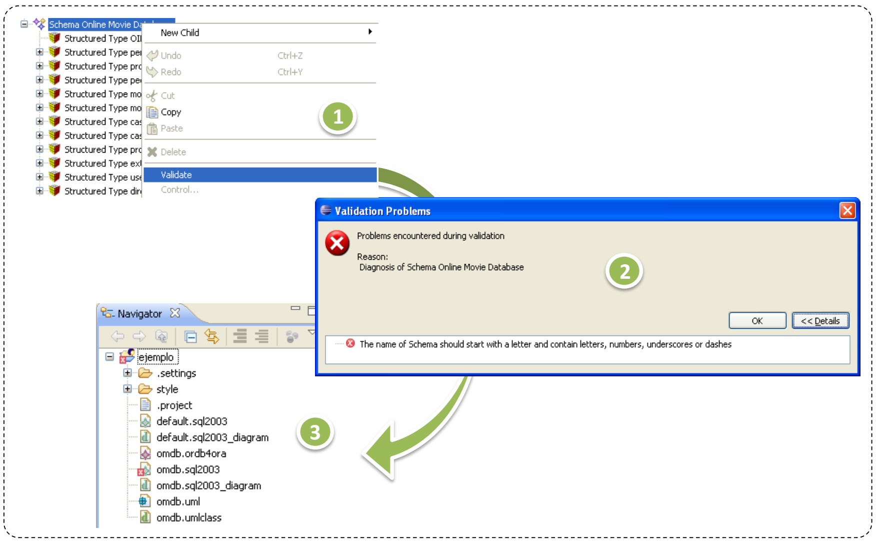
Extension Element Details

Set the properties of "constraintsBinding". Required fields are denoted by "*".

namespaceURI*:	http://SQL2003.ecore
constraints*:	Metamodels/SQL2003.evl
modelName:	
bundleId:	

Figure 5-68. Declaring extensions to the Epsilon validation plug-in

Since we have implemented validation in batch mode, the validation has to be invoked over the desired model as shown in Figure 5-69 (1). If any invariant does not evaluates to true, the message specified in those invariants are shown. In this case, the Schema name does not satisfy the constraits imposed (2), thus the model is marked as an invalid model (3).



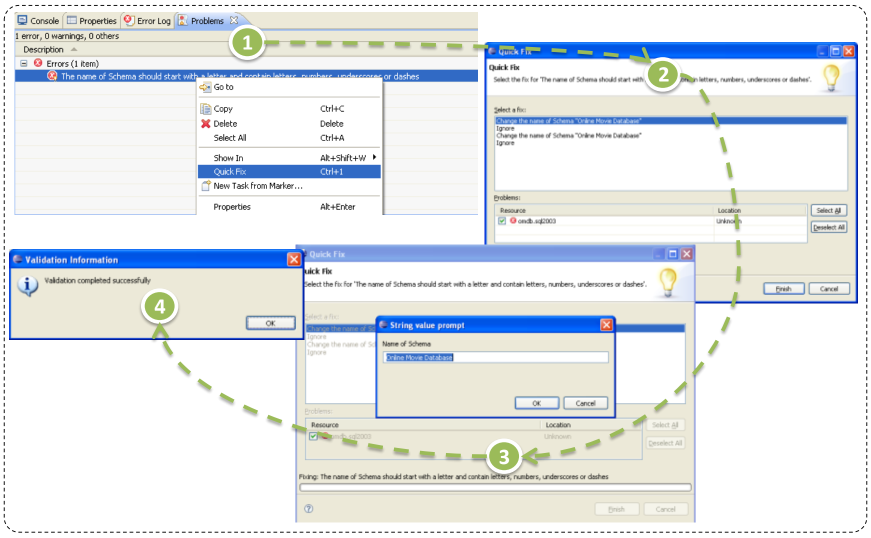
Problems encountered during validation

Reason:
Diagnosis of Schema Online Movie Database

The name of Schema should start with a letter and contain letters, numbers, underscores or dashes

Figure 5-69. Launching model validation in M2DAT.

Then, the problems view (see Figure 5-70) allows invoking the fixing behaviour coded in the .evl file (1).



1 error, 0 warnings, 0 others

Errors (1 item)
The name of Schema should start with a letter and contain letters, numbers, underscores or dashes

Select the fix for 'The name of Schema should start with a letter and contain letters, numbers, underscores or dashes'.

Change the name of Schema "Online Movie Database"
Ignore
Change the name of Schema "Online Movie Database"
Ignore

omdb.sql2003 Unknown

Validation completed successfully

Name of Schema
Online Movie Database

Fixing: The name of Schema should start with a letter and contain letters, numbers, underscores or dashes

Figure 5-70. Fixing validation problems

The available solutions are shown in a new window (2). In this case, the user might just Ignore the problem or ‘Change the name of Schema ---‘. If the latter is chosen, an input box let the user enter the new name for the Schema (3). Now, if validation is invoked again it raises a satisfactory evaluation.

5.6 Integrating New Modules in M2DAT

We have already mentioned a number of times that usability is a crucial aspect when developing tools for MDSD. The use of Eclipse helps in this sense since it provides with a common interface, devised to be extended and customized according to specific needs.

Regarding M2DAT, the main functionality that the user interface should provide is the way to invoke or launch, in a friendly way, the different model transformations (both model-to-model and model-to-text) bundled in each M2DAT’s module. To that end, this section presents the way we have used Eclipse’s facilities to develop model transformation launchers and to incorporate the needed controls in Eclipse that allows invoking such launchers.

We aim at showing that, developing such launchers and controls is feasible using existing components and available documentation, thus we will focus on presenting the results without going deep into the code that implement them. In this sense, readers interested are referred to the enclosed CD, that includes M2DAT-DB source code. Therefore, in the following we present some of the Eclipse extensions developed to build M2DAT-DB user interface.

5.6.1 Developing an Integration plug-in

In essence, the integration of the functionality provided by a new module in M2DAT resides in the development of an integration plug-in. Such plug-in implements the launchers for the model transformations bundled in the module, plus the add-ins for the user interface that invokes such launchers. Such plug-in depends on the differentplug-ins tha implement the DSLs bundled in the module and the plug-ins provided by the EMP that are used by the module.

For instance, Figure 5-71 shows the dependencies among the transformations bundled in M2DAT-DB and the different plug-ins that compose or uses the module.

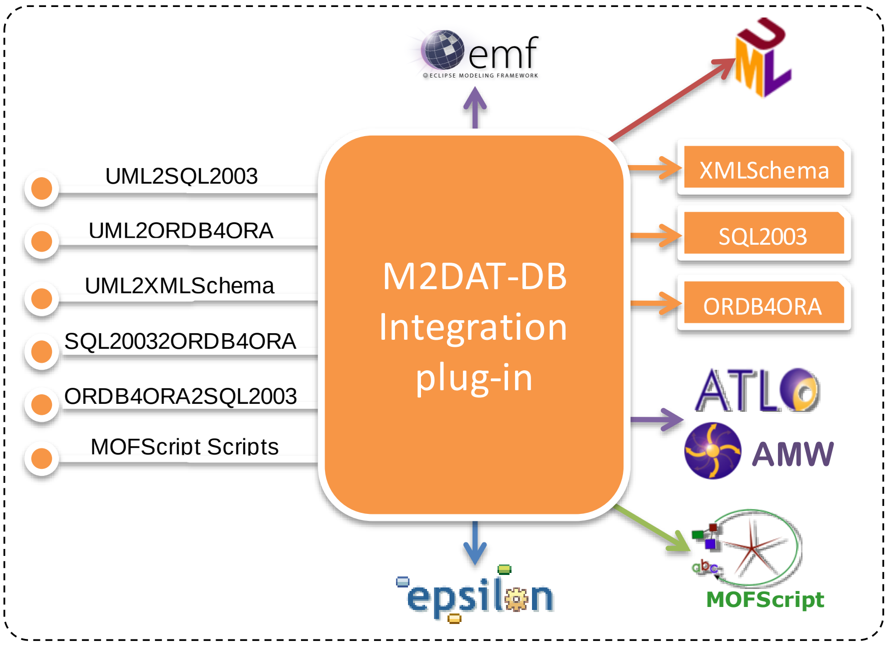


Figure 5-71. Dependencies between M2DAT-DB plug-ins and transformations

The integration plug-in for M2DAT-DB bundles five model-to-model transformations, plus three model-to-text transformations (the latter represented by the common moniker MOFScript Scripts). In addition, it depends on the different components provided by the EMP, like EMF, ATL, AMW, Epsilon and MOFScript, and the plug-ins that implement the three DSLs bundled in M2DAT-DB: XMLSchema, SQL2003 and ORDB4ORA.

In the following we describe the integration plug-in distinguishing the part supporting the launch of model transformations programmatically and the part that provides with a user interface to do it. Note that the latter leans on the former.

5.6.2 Launching Model Transformations Programmatically

To have an idea of what is needed in order to launch a model transformation, check the header of the UML2SQL2003 ATL transformation shown in Figure 5-72. It defines which are the source and target models and the libraries used by the transformation.

```
-- @atlcompiler atl2006
-- @nsURI              UML=http://www.eclipse.org/uml2/2.1.0/UML
-- @nsURI              SQL2003=http://SQL2003.ecore
-- @path               AMW=/UML2SQL2003/Metamodels/ORAnnotationMeta.ecore

module UML2SQL2003;
create OUT : SQL2003 from IN : UML, ANNOTATIONS : AMW;


-- IMPORTS        -------------------------
uses UML2SQL2003_constants;
uses UML2SQL2003_helpers;
uses UML2SQL2003_AMW;
uses UML;
```

Figure 5-72. ATL Header UML2SQL2003

According to such header, from a model conforming to the *UML* metamodel and a model conforming to the *AMW* metamodel, the transformation generates a model conforming to the *SQL2003* metamodel. At development time, the correspondences between such variables and real files or models is defined using the wizards provided by the ATL IDE. This way, the user creates a transformation execution that can be retrieved an reused at any moment. What we aim to do is to eliminate the need of having to do such execution configurations, or at least, simplify it, in order to ease the task and provide with more user-friendly interface for M2DAT. The first thing to do is to be able to launch the transformation programmatically, i.e. to configure the transformation execution programmatically. Once the configuration has been created it could be invoked from the code that handles the user interface events.

The main part of the the integration plug-in concerning the programmatic launch of model transformations in M2DAT's modules is a class called *Transformations*.

One of the main responsabilities of such class is to load the different transformations. Since this is a costly task in terms of memory and processing time, the *Transformations* class follows the singleton pattern to avoid replicating the load of metamodel. This way, the transformations will be loaded the first time the plug-in is used and remain loaded until Eclipse is closed. Figure 5-73 shows the beginning of M2DAT-DB's *Transformations* constructor.

```
private Transformations() {

modelHandler = (AtlEMFModelHandler)
            AtlModelHandler.getDefault(AtlModelHandler.AMH_EMF);

UML2ORDB4ORA_TransfoResource = Transformations.class.getResource
                        ("resources/UML2ORDB4ORA/UML2ORDB4ORA.asm");
UML2SQL2003_TransfoResource = Transformations.class.getResource
                        ("resources/UML2SQL2003/UML2SQL2003.asm");
UML2XMLSCHEMA_TransfoResource = Transformations.class.getResource
                        ("resources/UML2XMLSCHEMA/UML2XMLW.asm");
SQL2003ORDB4ORA_TransfoResource =  Transformations.class.getResource
                        ("resources/SQL2003ORDB4ORA/SQL2003ORDB4ORA.asm");
ORDB4ORA2SQL2003_TransfoResource =  Transformations.class.getResource
                        ("resources/ORDB4ORA2SQL2003/ORDB4ORA2SQL2003.asm");
}
```

Figure 5-73. Excerpt from M2DAT-DB's *Transformations* constructor: metamodels loading

In addition, the *Transformations* class contains a method to launch each model-to-model transformation bundled in the module. In contrast, all the model-to-text transformation launchers are encoded in an unique method (so-called *mofscriptTransformation*) since the configuration of MOFScript transformations is much more simpler that that of ATL transformations.

As an example, we will focus on the code to launch the UML2SQL2003 transformation, already mentioned a number of times along this dissertation. However, note that all the code that implements M2DAT-DB, and thus the integration plug-in, can be found in the enclosed CD.

Launching the UML2SQL2003 transformation programmatically

To be able to launch the UML2SQL2003 ATL transformation, the *Transformations* class from the M2DAT-DB's integration plug-in contains the *uml2sql2003* method, whose signature is displayed in Figure 5-74. Note that it receives three different parameters that correspond to the source and target models handled by the transformation.

```
public void uml2sql2003(String inUMLFilePath,String inAMWFilePath,
            String outFilePath) {
try {
            Map<String, Object> models = new HashMap<String, Object>();
            Map<String, Object> libraries = new HashMap<String, Object>();
            initSQLMetamodels(models);
            initSQLLibraries(libraries);
```

Figure 5-74. Signature of the UML2SQL2003 launcher

The first thing to do is to define a couple of has tables that will collect all the information provided by the ATL header to launch the transformation. In particular, which are the metamodels to use (*models*), as well as the libraries to import (*libraries*), in case there are some libraries to import. Next, such tables are populated by invoking the corresponding method, *initSQLMetamodels* (see Figure 5-75) and *initSQLLibraries* respectively.

```
private void initSQLMetamodels(Map<String, Object> models) {

   umlMetamodel = (ASMEMFModel) modelHandler.loadModel(
           "UML", modelHandler.getMof(),
           this.getClass().getResourceAsStream("resources/UML.ecore"));
   amwMetamodel = (ASMEMFModel) modelHandler.loadModel
           ("AMW", modelHandler.getMof(),
           this.getClass().getResourceAsStream
           ("resources/ORAnnotationMeta.ecore"));
   sql2003Metamodel = (ASMEMFModel) modelHandler.loadModel("SQL2003",
           modelHandler.getMof(),
           this.getClass().getResourceAsStream("resources/SQL2003.ecore"));

   models.put("UML", umlMetamodel);
   models.put("AMW", amwMetamodel);
   models.put("SQL2003", sql2003Metamodel);
}
```

Figure 5-75. *initSQLMetamodels* method

To that end, the ATL API for model handling is used. In particular, note the use of the *loadModel* method to recover each (meta)model by providing its path. This way, the keys in the hash table corresponds with the variable names used in the ATL header ("UML", "AMW" and "SQL2003"), whereas the values correspond to the respective (meta)models.

Next thing to do is to load the models handled by the transformation in the *models* hash table. Figure 5-76 shows the corresponding code. Again, the ATL API is used to that purpose.

```
// get/create models
ASMEMFModel umlInputModel = (ASMEMFModel) modelHandler.loadModel
          ("IN", umlMetamodel,  URI.createFileURI(inUMLFilePath));
models.put("IN", umlInputModel);

if(inAMWFilePath != null)
{
          ASMEMFModel amwInputModel = (ASMEMFModel) modelHandler.loadModel
                    ("amw", amwMetamodel, URI.createFileURI(inAMWFilePath));
          models.put("amw", amwInputModel);
}

ASMEMFModel orOutputModel = (ASMEMFModel) modelHandler.newModel("OUT",
          URI.createFileURI(outFilePath).toFileString(), sql2003Metamodel);
models.put("OUT", orOutputModel);
```

Figure 5-76. Loading models for executing an ATL transformation

Finally, once all the models and metamodels have been loaded, it is time to execute the transformation, using once more, the methos provided by the ATL API.

```
// launch
AtlLauncher.getDefault().launch(this.UML2SQL2003_TransfoResource,
          libraries, models, Collections.EMPTY_MAP,
          Collections.EMPTY_LIST, Collections.EMPTY_MAP);

modelHandler.saveModel(orOutputModel, outFilePath, false);
```

Figure 5-77. Launching an ATL transformation programmatically

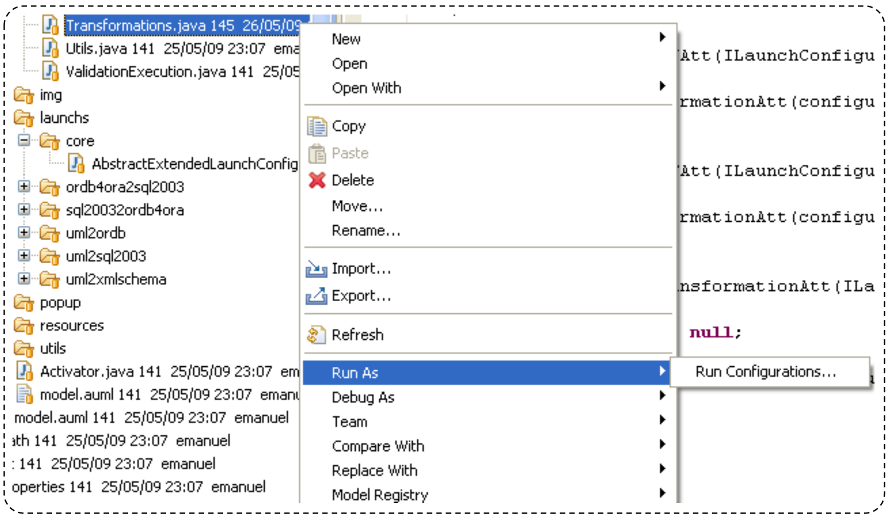
As a result, the target model is stored in the resource pointed by the *outFilePath*.

5.6.3 Adding Graphical Support for launching Model Transformations

Once we are able to launch a model transformation programmatically, it is time to develop the support to be able to invoke it from the user interface. To that end, Eclipse provides with some kind of generic launchers that can be extended according to specific needs. It is based on two main concepts: **Launch Configurations** (aka as Run Configurations) and **Launch Configuration Types**.

At the simplest level, **LaunchConfigurationTypes** are cookie cutters, and **LaunchConfigurations** are the cookies made from these cookie cutters. When a plug-in developer decides to create a launcher, what he is really doing is creating a specific kind of cookie cutter that will allow users to stamp out as many cookies as they need. In slightly more technical terms, a LaunchConfigurationType (henceforth, a 'config type') is an entity that knows how to launch certain types of launch configurations, and determines what the user-specifiable parameters to such a launch may be. Launch configurations (henceforth, 'configs') are entities that contain all information necessary to perform a specific launch. For example, a config to launch a HelloWorld Java application would contain the name of the main class ('HelloWorld'), the JRE to use (JDK1.4.1, for example), any program or VM arguments, the classpath to use and so on. When a config is said to be 'of type local Java Application', this means that the local Java application cookie cutter was used to make this config and that only this config type knows how to make sense of this config and how to launch it [339].

For instance, Figure 5-78 shows that the contextual menu of any file in the Eclipse workspace gives access to the different Run Configurations that might be launched using such file as input.


Transformations.java 145 26/05/09
Utils.java 141 25/05/09 23:07 ema
ValidationExecution.java 141 25/05
img
launchs
core
AbstractExtendedLaunchConfig
ordb4ora2sql2003
sql20032ordb4ora
uml2ordb
uml2sql2003
uml2xmlschema
popup
resources
utils
Activator.java 141 25/05/09 23:07 em
model.auml 141 25/05/09 23:07 emanu
model.auml 141 25/05/09 23:07 emanuel
ath 141 25/05/09 23:07 emanuel
: 141 25/05/09 23:07 emanuel
operties 141 25/05/09 23:07 emanuel
New
Open
Open With
Copy
Paste
Delete
Move...
Rename...
Import...
Export...
Refresh
Run As
Debug As
Team
Compare With
Replace With
Model Registry
Run Configurations...

```
Att(ILaunchConfigu
rmationAtt(configu
Att(ILaunchConfigu
rmationAtt(configu
nsformationAtt(ILa
 null;
```

Figure 5-78. Eclipse's shortcut to Run Configurations

In order to add graphical support for launching model transformations, M2DAT-DB uses such APIs and mechanisms to build a graphical wrapper for the programmatic launchers commented in the previous section. This wrapper incude two big groups of controls:

- First, M2DAT-DB includes five new Launch Configuration Types, one for each model-to-model transformation bundled in M2DAT-DB.
- Second, M2DAT-DB adds actions to the contextual menus of the models produced by M2DAT-DB to invoke the different model-to-text transformations. We have proceed this way because model-to-text transformations are much more simpler to launch since they require less parameters. In essence, all the information needed to execute the model transformations is in the source model (i.e. the file over which the execution of the model transformation is invoked). Therefore, there is no need to create a new Launch configuration type to that purpose or to bother the user with wizards full of controls to fill.

In the following we show some results. Nevertheless, remember that the code of M2DAT-DB can be found on the CD enclosed.

5.6.3.1 Launch Configuration Types for M2DAT-DB model-to-model transformations

As Figure 5-79 shows, M2DAT-DB includes five new types of Run Configurations, one for each model-to-model transformation supported.

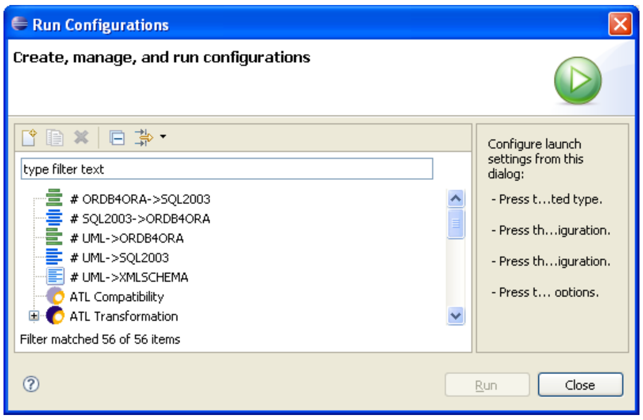
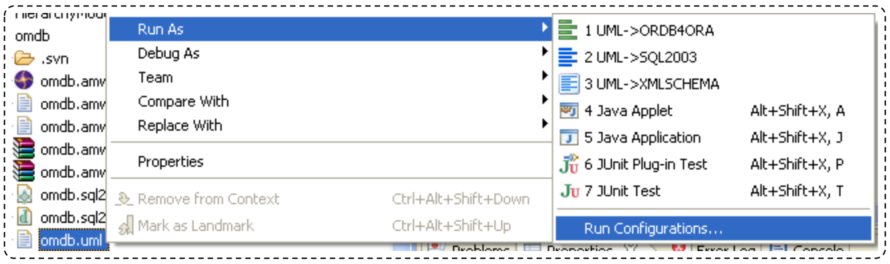


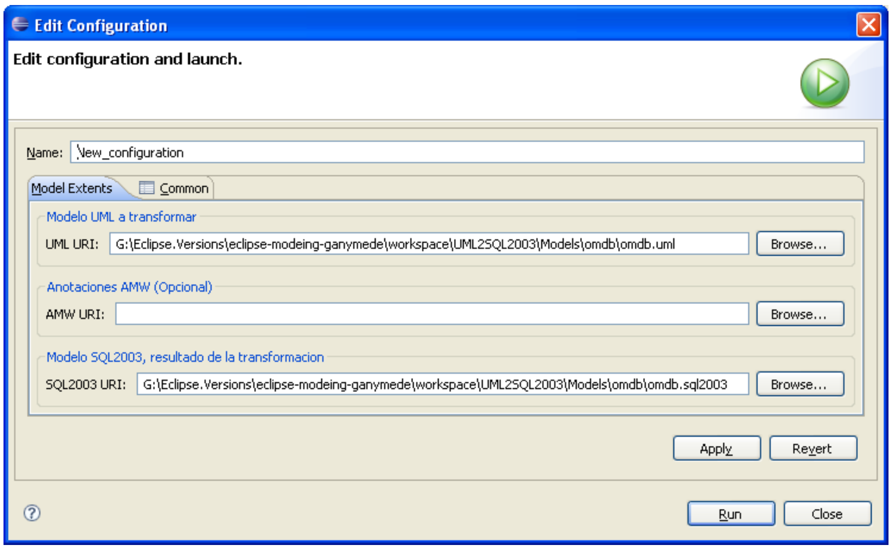
Figure 5-79. M2DAT-DB's Run Configuration Types

After creating a new Run Configuration Type, it can be used in any view of the Eclipse workspace. For instance, Figure 5-80 shows that the contextual menu shown for UML models includes the ability to access the new Run Configuration Types that are appropiate for UML models. That is, UML->ORDB4ORA, UML->SQL2003 and UML->XMLSCHEMA.


omdb
.svn
omdb.amw
omdb.amw
omdb.amw
omdb.amw
omdb.amw
omdb.sql2
omdb.sql2
omdb.uml
Run As
Debug As
Team
Compare With
Replace With
Properties
Remove from Context Ctrl+Alt+Shift+Down
Mark as Landmark Ctrl+Alt+Shift+Up
1 UML->ORDB4ORA
2 UML->SQL2003
3 UML->XMLSCHEMA
4 Java Applet Alt+Shift+X, A
5 Java Application Alt+Shift+X, J
6 JUnit Plug-in Test Alt+Shift+X, P
7 JUnit Test Alt+Shift+X, T
Run Configurations...
Figure 5-80. M2DAT-DB's Run Configuration shortcuts

If the user clicks over the second one, the wizard to define UML->SQL2003 Run Configurations is shown (see Figure 5-81).

Then, the user can use the wizard to provide with the parameters needed to execute the transformation, i.e. the source models and where to store the target model. Note that it already recognises one of the source models (the one over the contextual menu was invoked) and provides with a tentative location for the target model. By contrats, no default annotation model is provided since annotating the conceptual data model (the UML model) is not mandatory. Note also that the user does not need to specify the location of the metamodels and the ATL libraries used by the transformation. The result is quite user-friendly.


Edit Configuration

Edit configuration and launch.

Name: New_configuration

Model Extents | Common

Modelo UML a transformar

UML URI: G:\Eclipse.Versions\eclipse-modeing-ganymede\workspace\UML2SQL2003\Models\omdb\omdb.uml

Anotaciones AMW (Opcional)

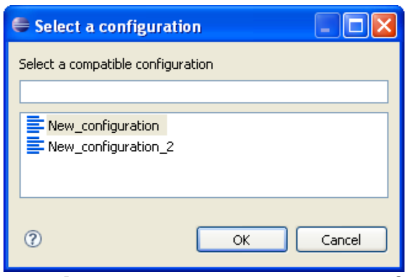
AMW URI:

Modelo SQL2003, resultado de la transformacion

SQL2003 URI: G:\Eclipse.Versions\eclipse-modeing-ganymede\workspace\UML2SQL2003\Models\omdb\omdb.sql2003

Figure 5-81. UML2SQL2003 Run Configuration Wizard

In addition, such configuration is automatically stored with the name provided in the Name field. Thus, the user can invoke the same transformation execution as many times as needed, without the need to configure it again. For instance, if the user has defined a couple of UML->SQL2003 Run Configurations, whenever he uses the shortcut over the same file,the window in Figure 5-82 will be shown in order to let him choose which is the Configuration he wants to run.


Select a configuration

Select a compatible configuration

New_configuration
New_configuration_2

Figure 5-82. Selecting a UML2SQL2003 RunConfiguration

5.6.3.2 Shortcut menus and Contributing Actions for M2DAT-DB model-to-text transformations

As mentioned before, the execution of model-to-text transformations needs from less information or parameters that the one of model-to-model transformations. Therefore, the user interface to launch model-to-text transformations could be limited to the addition of some controls that allow launching the transformation. Indeed, no extra information, apart from the source model, is needed to execute the transformation.

This way, M2DAT-DB includes controls to launch the different model-to-text transformations that supports. All of them have been developed following the process sketched in [19]. The controls developed can be divided in two main groups: shortcut menus and contributing actions.

Regarding the former, the contextual menu shown over any type of M2DAT-DB model, i.e. ORDB4ORA, SQL2003 or XMLSchema, includes a shortcut to generate the corresponding code from the given model. For instance, Figure 5-83 shows the contextual menu for a .sql2003 file.

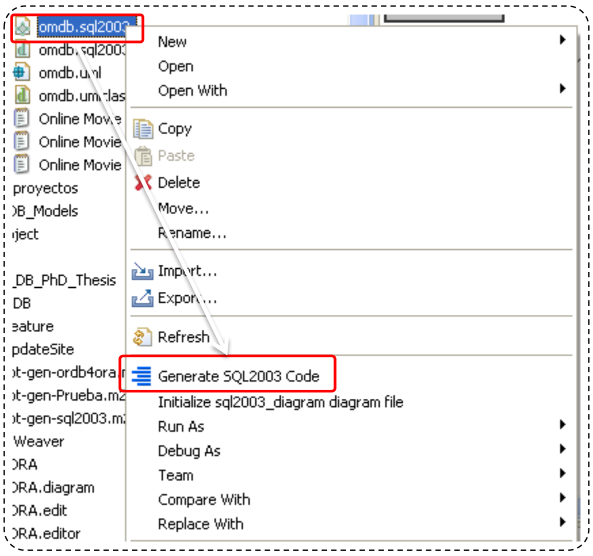


Figure 5-83. SQL2003 Shortcut Menu

On the other hand, M2DAT-DB contributes the Navigator view of Eclipse's workspace with new actions to invoke the model-to-text transformations bundled in the module. This way, whenever the user selects one M2DAT-DB model, the corresponding action becomes active and can be invoked by the user, while the rest of time is remains shadowed. Figure 5-84 shows the result.

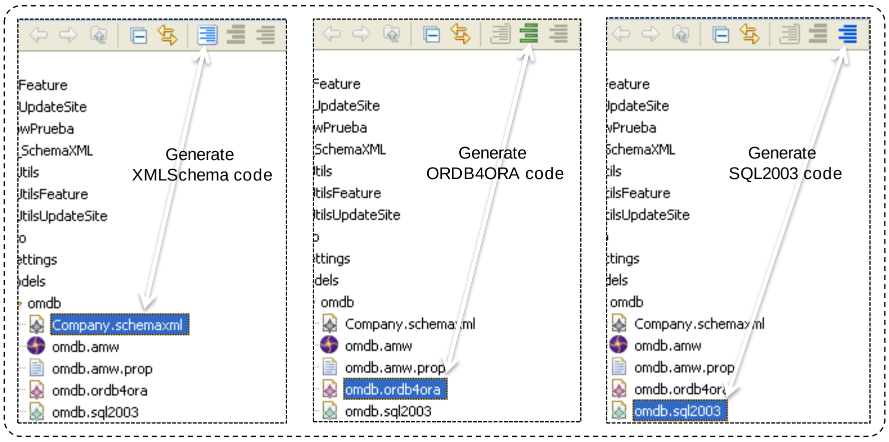


Figure 5-84. M2DAT-DB's Actions contributed to Eclipse Navigator's toolbar

Depending on the type of file the user selects, the corresponding control becomes active, allowing the user to launch the model-to-text transformation that generates code from the model selected.

Conclusion

To conclude this dissertation, this chapter summarizes the main contributions of this thesis and contrasts the fulfilled objectives with those stated at the beginning of the thesis. In addition, an analysis of the results is provided next to the enumeration of the publications that serve to contrast them both on national and international forums. Besides, a number of questions for further research are raisen next to the directions to follow in order to tackle them.

6.1 Analysis of Achievements

At the beginning of this dissertation, section 1.2 stated a set of partial objectives to fulfil the main objective if this thesis: the specification of a technical solution for the construction of a framework to support model-driven development of Web Information Systems.

In the following, the achievement of those objectives is analysed:

O1. Analysis and evaluation of existing tools for MDE tasks in order to identify the most suitable to build a framework for model-driven development of Web Information Systems.

To fulfil this objective, chapter 2 provided, first, with a detailed review of existing solutions to build the support for model-driven methodological proposals according to a set of relevant features convenient for developing M2DAT. Therefore, the review was focused on existing MDE technology to build an open-source framework that promotes extensibility and interoperability.

The main conclusions gathered from that review spin around the selection of technologies in the context of the Eclipse Modelling Project for building M2DAT. This decision promotes extensibility and interoperability. In particular, the Eclipse Modelling Framework was selected as metamodelling technology and underlying basis for M2DAT. It provides with the basic capabilities for defining new DSLs, a basic toolkit to work with the new DSLs and, what is more relevant, the extension mechanisms needed to adequate the basic generated toolkit to the specific needs of a particular methodology. In addition, the core metamodel, Ecore, is becoming the de-facto standard for metamodelling tasks, thus using EMF maximizes the interoperability of any tool develop atop of it.

O2. Analysis and evaluation of existing frameworks for MDSD.

The previous objective was set with the idea of identifying the most convenient tools to build the tooling for supporting a MDSD methodology. In contrast, this objective was set to asses the main drawbacks of existing works in the field. Therefore, Chapter 2 provided with a complete analysis and evaluation of existing frameworks for model-driven development of software for concrete domains.

On the one hand, regarding existing tools supporting methodologies for model-driven development of Web Information Systems, the main findings had to do with the lack of interoperability and extensibility. As it has been described along this dissertation, M2DAT has solved the shortcomings detected in previous works in this sense.

On the other hand, before building M2DAT-DB a review of existing works for model-driven development of modern DB schemas was also performed. The conclusions gathered confirmed that there are no frameworks that support model-driven development of Object-Relational Database Schemas while model-driven development of XML Schemas is just partially supported since there are no solution supporting both PIM and PSM models. M2DAT-DB, the reference implementation for M2DAT developed as part of this thesis fills the aforementioned gaps.

O3. Specification of the conceptual architecture of M2DAT framework.

The conceptual architecture of M2DAT, presented in section 4.1 has been defined starting from the architecture of MIDAS-CASE (section 3.2). MIDAS-CASE was a first step towards the complete specification of the MDSD framework presented. Its architecture was thought as a set of co-existing modules or subsystems, each one providing with specific capabilities. The underlying idea was to encapsulate all the functionality related with a given concern of the system in just one place. Therefore, when new concerns were to be considered for the development of the system, a new module was to be developed and integrated with the rest of modules.

M2DAT's architecture follows the same approach and is structured according to two orthogonal dimensions:

- On the one hand, M2DAT can be thought of as a set of modules, one for each concern of the system development. It encapsulates a set of DSLs to model, at different abstraction levels, the set of concepts related with such concern, plus the model transformations that bridge them.

- On the other hand, M2DAT's architecture follows the classical separation between the interface and the application logic plus the persistence layer. The **interface** is composed mainly of one or more graphical editors for each model and the wizards needed to integrate them. The **application logic** is encoded in a common module so-called **model processor** that encapsulates model transformation, model validation, code generation and the like.

O4. Selection of the technologies to be used for M2DAT

Chapter 4 gave an overview on the design decisions that drove the mapping from the conceptual architecture of M2DAT to a technical design. In other words, which are the approaches and technologies adopted. To that purpose, a set of discussions around the most adopted ways of addressing the main tasks related with deploying model-driven software development proposals were presented.

At the end of each section, the selected option for M2DAT was described along with the criteria used to justify such decision. In other cases where there were no space for selection since the right way of performing the task (in a model-driven context) was one and unique, such way of working has been described, as well as its uniqueness justified.

For instance, new modelling languages are typically defined following two different approaches. Such dichotomy between UML profiles or DSLs was tackled in section 4.2.

Regarding model-to-model transformation languages, the hybrid approach was selected as the most convenient, emphasizing the relational style since it fits better with the declarative nature of model transformations. In particular, after several tests with different languages, the ATLAS Transformation Language was chosen as model-to-model transformation language.

Finally, model-to-text transformation has been tackled with MOFScript, so far, this remains an open issue. In fact, the OMG standard for this task is still quite recent [266], what proves that there is still lot of space for improvement on this field.

O5. Specification of the technical design of M2DAT.

The set of tools for MDE tasks selected to implement M2DAT are integrated under the common architecture sketched in section 4.1.2. In essence, this design is the summary of the methodlogical and technlological decisions that are presented and justified along the rest of the sections of chapter 4.

O6. Specification of the development process for each M2DAT module.

As it has been clearly stated along this dissertation, this thesis will serve as a basis for future research works. It lays the foundations to develop the technological support for forthcoming advances on model-driven development of information systems that MIDAS methodlogy will incorporate as long as they appear. Thereby, as soon as new concerns are to be considered for the development of a system according to MIDAS methodology, the corresponding technical support will be developed. To that purpose, section 4.8 described the development process to follow when building new M2DAT modules. Basically, it combines the techniques and technical solutions previously presented along that same chapter.

O7. Validation of M2DAT specification

The conceptual architecture and the technical design of M2DAT, next to the proposed development process for new modules constitute the specification of M2DAT. However, a specification is unuseful without a reference implementation showing its feasibility and how it is to be interpreted. Thereby, a proof of concept for M2DAT has been provided by building M2DAT-DB, a M2DAT module that supports model-driven development of modern database schemas.

Chapter 5 focused on showing how the technical design of M2DAT and the development process proposed were applied to build M2DAT-DB. Besides, the result of building M2DAT-DB was shown. This way, the construction of M2DAT-DB served as reference implementation for the specification of M2DAT specification. It confirmed that the proposal is implementable and it has clarified how the specification of the architecture and the development process has to be interpreted.

Besides, a set of case studies served to complete the validation of the proposal. To that end, a complete set of case studies performed with M2DAT-DB, the reference implementation for M2DAT, were described. Such case studies contributed on the detection of errors and improvements for M2DAT-DB that served to refineM2DAT specification.

6.2 Main Contributions

This thesis has resulted in a number of contributions, regarding not only the scope of this research (M2DAT specification) but also related with other collateral aspects. Some of them were objectives fixed before addressing this work while others have emerged during its development. They are summarized in the following.

A complete analysis of the existing solutions for building frameworks supporting methodologies for model-driven development of Web Information Systems.

The technical decisions that drive the open specification of M2DAT have been clearly justified along this dissertation. However, on the way to such decisions a complete review of existing technology in the field of MDE has been provided. This review is in its turn a clear contribution of this thesis since it could be used for more specific purposes.

Some of the reviewing frameworks for model-driven development of software are being improved (or it is planned to do so) in order to adapt them to advances in the field. The findings and analysis provided in the state of the art of this thesis might help the researchers behind those works in the selection of the technology that best suit their needs (that not necessarily have to be the same of M2DAT).

For instance, model transformation is a field where those frameworks admit a lot of improvement. In this sense, this thesis provides with a number of valuable conclusions and lessons learned in order to bring the advantages of current technology to existing frameworks, like the use of annotation models or the drawbacks related with using pure imperative approaches for model transformation development.

Specification of a framework for model-driven development of Web Information Systems.

The main contribution of this thesis has been the specification of M2DAT, an open MDSD framework that supports model-driven development of Web Information Systems. The main contributions of M2DAT regarding previous works are its **extensible** and **interoperable nature** and the support for customizable transformations by means of annotation models.

These features contribute to simplify the extension of the framework in order to support new capabilities. Whenever the underlying methodology, MIDAS, is to be extended by adding a new concern to MIDAS architecture, a new module will be built atop of M2DAT. Such module will support the corresponding method. Besides, the models supported by the new module will be directly interoperable with existing models with no extra effort, i.e. without the need of building technological bridges between technical spaces [208] as it has been traditionally done. Indeeed, though any modelling tool might be said to be located in the same technological space that the rest, this is not completely true when it comes to implementation. Importing/Exporting models to/from one tool to/from the other implies moving through different technical spaces, typically grammarware and modelware. Though there are technical solutions for these tasks, it usually entails some loss of semantics and it is always prone to errors. In contrast, when the models are defined atop a common metametamodel, the only artefact needed to bridge them is a model transformation and, optionally a weaving model to drive the mapping.

On the other hand, none of the previous works offered support for **customizable transformations**. This way, when the processes were completely automated, the only way of having some control over the resulting system was modifying the models handled. Even after modifying the models, some of the model transformations bundled on those frameworks were never able to produce some constructions on the target models. The only way of including those constructions on the models (thus, of generating the code that implements them in the system) was to refine the output model by hand. In constrast, M2DAT model transformations consider the use of annotation models to drive the execution of the transformation. This way, any construction might be obtained on the target model without decreasing the level of automation. At the same time, the design decisions that contribute to produce the specific target model are persisted. Moreover, the software artefact containing these decisions is the most natural in a MDE environtment: another model.

Furthermore, the specification of M2DAT comprises the definition of the **development process** to follow in order **to build M2DAT modules** as well as a the **reference implementation**, M2DAT-DB, that clarifies the way the specification is to be used and shows the result of doing so.

Support for semi-automatic model-driven development of modern DB schemas.

Although the objective of developing M2DAT-DB was mainly providing with a proof of concept for M2DAT proposal, it constitutes a complete and open framework for model-driven development of modern database schemas. The state of the art showed that there were no previous works providing support for this task. Hence, M2DAT-DB itself is a contribution of this thesis.

To that end, M2DAT-DB provides with a DSL toolkit for developing OR DB schemas conforming to the SQL:2003 standard and OR DB schemas for Oracle. Such toolkit supports the generation of the OR DB schema from a UML2 conceptual data model. Besides, the generation process might be customized by attaching an annotation model to the conceptual data model (though it is not mandatory since a default generation is provided). Besides, it bundles the model transformations needed to move from the OR model for SQL:2003 to the one for Oracle and the other way round. Likewise, graphical editors are provided to handle Oracle and SQL:2003 models.

To complete the support, M2DAT-DB also bundles a toolkit for development of XML Schemas following the same approach. That is, the XSD model is obtained from a conceptual data model and the generation process lend space to the introduction of design decisions by means of an annotation model.

6.3 Scientific Results

Some of the results of this thesis have been published in different fourms, both national and international. In the following, those publications are grouped according to the type of publication.

- **Articles in International Journals**
    - Vara, J.M., De Castro, V., Didonet Del Fabro, M. & Marcos, E. (2009). Using Weaving Models to automate Model-Driven Web Engineering proposals. *International Journal of Computer Applications in Technology*. (Accepted to be published)
    - Koch, N., Meliá, S., Moreno, N., Pelechano, V., Sánchez, F. & Vara, J.M. (2008). Model-Driven Web Engineering. *UPGRADE, IX*(2), 40-45. (April 2008)
    - Vara, J.M., De Castro, V. & Marcos, E. (2005). WSDL Automatic Generation from UML Models in a MDA Framework. *International Journal of Web Services Practices, 1*(1-2), 1-12.
- **Articles in Iberoamerican Journals**
    - Vara, J.M., Vela, B., Cavero, J. M. & Marcos, E. (2007). Transformación de Modelos para el Desarrollo de Bases de Datos Objeto-Relacionales. *IEEE Latin America Transactions, 5*(4), 251-258. (July 2007)
- **Articles in National Journals**
    - Vara, J M., Acuña, C. J., Marcos, E. & Lopez Sanz, M. (2004). Desarrollo de un Sistema de Información web: una experiencia con Oracle XMLDB. *CUORE (Círculo de Usuarios de Oracle España), VIVAT ACADEMIA, 27*, 3-12.

- **Articles in International Conferences**
  - López Sanz, M. Vara, J.M., Marcos, E. & Cuesta, C. A Model-Driven Approach to Weave Architectural Styles into Service-Oriented Architectures. 1st International Workshop on Model-Driven Service Engineering (MoSE 2009). Hong Kong, China. (November 6, 2009). (Accepted to be published).
  - Vara, J.M., Vela, B., Bollati, V. & Marcos, E. (2009). *Supporting Model-Driven Development of Object-Relational Database Schemas: a Case Study*. **ICMT2009 - International Conference on Model Transformation**, Zurich, Switzerland. (29-30 June, 2009) (**Acceptance Ratio: 22%**).
  - Vara, J.M., Bollati, V., Vela, B. & Marcos, E. (2009). *Leveraging Model Transformations by means of Annotation Models*. 1st International Workshop in Model Transformation with ATL (MtATL 2009), Nantes, France. (July 8-9, 2009).
  - Vara, J.M., Didonet Del Fabro, M., Jouault, F. & Bézivin, J. (2008, 29/01/2008). *Model Weaving Support for Migrating Software Artefacts from AUTOSAR 2.0 to AUTOSAR 2.X.* 4th European Congress on EMBEDDED REAL TIME SOFTWARE (ERTS 2008), Toulouse, France. (January 29- 31, February 1, 2008).
  - Vara, J.M., De Castro, V. & Marcos, E. *From Real Computational Independent Models to Information System Models: an MDE approach*. Proc. of the 4th International Workshop on Model-Driven Web Engineering (MDWE 2008), Toulosue, France. (September 30, 2008). CEUR Workshop Proceedings, ISSN 1613-0073.
  - Vara, J.M., Vela, B., Cavero, J.M. & Marcos, E. (2007). *Model Transformations for Object-Relational Databse Development.* **ACM Symposium on Applied computing (SAC 2007)**, Seoul, Korea. ACM Press. (11-15 March, 2007). (**Acceptance Ratio: 32.5%**).
  - De Castro, V., Vara, J.M. & Marcos, E. *Model Transformation for Service-Oriented Web Applications Development*. Proc. of the 3rd International Workshop on Model-Driven Web Engineering (MDWE 2007), Como, Italy. (July 16-20, 2007). CEUR Workshop Proceedings, ISSN 1613-0073.
  - Caceres, P., De Castro, V., Vara, J.M. & Marcos, E. (2006). *Model Transformations for Hypertext Modelling on Web Information Systems.*

ACM Symposium on Applied computing (SAC 2006), Dijon, France. ACM Press. (23–27 April, 2006). (**Acceptance Ratio: 32%**).

- Vara, J.M., De Castro, V. & Marcos, E. (2005). *WSDL Automatic Generation from UML Models in a MDA Framework.* International Conference on Next Generation Web Services Practices (NWeSP), Seul, Korea. (22-26 August, 2005)IEEE Computer Society Press.

- **Articles in Iberoamerican Conferences**
  - Acula, C. Minoli, M. & Vara, J.M. *Model Driven Development of Semantic Web Services using Eclipse Modelling Languages*. 10th Mexican International Conference on Computer Science (ENC 2009). Mexico City, Mexico. (21-25 September, 2009). (Accepted to be published).
  - Bollati, V.A., Vara, J.M, Vela, Belén & Marcos, E. *Uso de Modelos de Anotación para automatizar el Desarrollo Dirigido por Modelos de Esquemas XML*. XII Conferencia Iberoamericana de Ingeniería de Requisitos y Ambientes de Software. (IDEAS'09), Medellín (Colombia). (Abril 13-17, 2009).
  - Bollati, V.A., Vela, B., Vara, J.M. & Marcos, E. *Una Aproximación Dirigida por Modelos para el Desarrollo de Bases de Datos Objeto-Relacionales*. XIV Congreso Argentino de Ciencias de la Computación. (CACIC 2008). Chilecito (La Rioja, Argentina). (October 6th-10th, 2008).
  - De Castro, V., Vara, J.M., Herrmann, E. & Marcos. *A Model Driven Approach for the Alignment of Business and Information Systems Model.* 9º Mexican International Conference on Computer Science (ENC 2008). Mexicali, Baja California, Mexico. (6-10 October, 2008). (Acceptance Ratio: 26%). IEEE Computer Society.
  - Bollati, V.A., Marcos, E., Vara, J.M. &. Vela, B. *Analisis de Herramientas MDA*. XIII Congreso Argentino de Ciencias de la Computación. (CACIC 2007). Corrientes and Resistencia, Argentina. (October 1st-5th, 2007)
  - Molina, F., Lucas, F. J., Toval, J. A., Vara, J.M. & Marcos, E. (2006). *Soporte CASE para el desarrollo preciso de Sistemas de Información WEB.* IADIS International Conference, WWW/Internet 2006, Murcia, Spain. (5-8 October, 2006)

- Vara, J.M., De Castro, V., Caceres, P. & Marcos, E. (2004). *Arquitectura de MIDAS-CASE: una herramienta para el desarrollo de SIW basada en MDA.* IV Jornadas Iberoamericanas en Ingeniería del Software e Ingeniería del Conocimiento. JIISIC'04, Madrid, Spain. (3-5 November, 2004). (Acceptance Ratio: 49%).

- **Articles in National Conferences**
  - Bollati, V.A., Vara, J.M., Vela, B. & Marcos, E. *Una Aproximación Dirigida por Modelos para el Desarrollo de Esquemas XML*. **XIII Jornadas de Ingeniería del Software y Bases de Datos (JISBD'08)**. Gijón, Spain. (October 7-10, 2008). (**Acceptance Ratio: 25%**)
  - Vara, J.M., Bollati, V., Vela, B. & Marcos, E. *Uso de Modelos de Anotación para automatizar el Desarrollo Dirigido por Modelos de Bases de Datos Objeto-Relacionales*. V Taller sobre Desarrollo de Software Dirigido por Modelos. DSDM'08. Gijón, Spain (España) (October 7, 2008).
  - Vara, J.M., De Castro, V., Didonet Del Fabro, M. & Marcos, E. (2008). *Using Weaving Models to automate Model-Driven Web Engineering proposals*. ZOCO'08: Integración de Aplicaciones Web. Gijón, Spain (España) (October 7, 2008).
  - De Castro, V., Vara, J.M., Herrmann, E. & Marcos, E. *Obteniendo Modelos Sistemas de Información a partir de Modelos de Negocios de Alto Nivel: Un Enfoque Dirigido por Modelos*. IV Jornadas Científico-Técnicas en Servicios Web y SOA (JSWEB'08). Sevilla, Spain (October 29-30, 2008).
  - Vara, J.M., Vela, B., Cavero, J.M. & Marcos, E. (2007). *Transformación de Modelos para el Desarrollo de Bases de Datos XML*. III Taller sobre Desarrollo Dirigido por Modelos. MDA y Aplicaciones (DSDM'06) - XI Jornadas de Ingeniería del Software y Bases de Datos, JISBD'2006. Sitges, Spain. (4 October, 2006).
  - Vara, J. M., Vela, B., Cavero, J.M. & Marcos, E. (2007). *Transformación de Modelos para el Desarrollo de Bases de Datos XML*. **XI Jornadas de Ingeniería del Software y Bases de Datos, JISBD'2006**. Sitges, Spain. (3-6 October, 2006). (**Acceptance Ratio: 35%**).
  - Vara, J.M., De Castro, V. & Marcos, E. (2005). *Generación Automática de WSDL a partir de Modelos UML*. I Jornadas Científico-Técnicas en Servicios Web (JSWEB 2005). Granada, Spain. (13-14 September, 2005).

- **Patents**
  - Title: *M2DAT/DB: Herramienta para el Desarrollo Dirigido por Modelos de BD*.
    - Inventors: E. Marcos, B. Vela, J.M. Vara, V. Bollati
    - Application Nº: M-8452/2008
    - Priority Country: España
    - Priority Date: 24/10/2008
    - Holder Entity: Universidad Rey Juan Carlos
    - Extended Countries: Spain

6.4 Future Work

Despite the contributions made on this thesis, it has detected several directions to further work. Some of them were just not considered as objectives of this thesis while others have emerged during the development of this work. In the following, we summarize some of them.

6.4.1 Development of M2DAT Modules

As we have mentioned a number of times along this dissertation, the main objective of this work has been the specification of M2DAT. Besides, we have built one M2DAT module (M2DAT-DB) in order to provide with a reference implementation for the specification.

Actually, this specification has been devised to be extensively applied during the next years when building the support for the rest of MIDAS methodology. Indeed, the main direction for further work of this thesis is exactly that: the development of the modules that will support the rest of MIDAS methodology attending to the specification of M2DAT provided in this thesis.

Likewise, notice that the construction of the whole framework is an endless task since MIDAS is open to include new views of the system. In fact, its modular architecture was devised to promote extensibility of the framework by inclusion of new views. Accordingly, M2DAT has to be also open to integrate support for them. Therefore, both the definition of the conceptual architecture of M2DAT and the decisions that have driven its technical design have been made with the aim of easying the integration of new modules.

In fact, we have already started to work on the technical support for the behaviour view [107, 108] or the semantics view [8]. Besides, the thesis from Marcos Lopez will incorporate support for the architecture and the one from Elisa Herrmann will improve code generation facilities.

6.4.2 Traceability in Model Transformations

Traceability has been always a relevant topic in Software Engineering. Maintaining the links from requirements forward to corresponding design artifacts, code, and test cases has attracted the attention of researchers for long time as a way of performing impact analysis, regression tests, requirements validation, etc. [27].

With the advent of MDE and the MDA, traceability management has even gained relevance. The key role of models as driving force in the development process eases the task of maintaining the traces from the requirements to the working-code. Indeed, the main artefacts obtained along the development process are models. Thus, handling traceability might be simplified to the creation and maintenance of traces between the elements of such models. Even more, such traces could be automatically generated if the models are connected by a model transformation and the language used offers support to keep information about which elements are related to which by the model transformation [342]. This way, if some element from the source model is modified, the modification might be replicated over the corresponding element of the target model. Actually, this is a mandatory feature according to the QVT standard though technical support is still quite inmature.

Besides, in the context of MDA traceability management deals also with CIM to PIM traceability. In MDA literature, little is said about the CIM-to-PIM mapping and MDA tools do not use to support it [165]. This fact is mainly due to the different nature of both models. The CIM serves to model the requirements for the system, describing the situation in which the system will be used [246]. In essence, it might be shown as the business/domain model [193]. By contrast, if the business or organization uses some kind of software system, it will be described in a specific model to that end, the PIM, which provides with a description of the software system. Business and software system are rather different. Therefore, automatic derivation of a PIM from a CIM is not always feasible. The (human) designer has to state which things from the CIM will be translated to a software system, and accordingly define the corresponding PIM for such sytem. At best, some information for the PIM might be extracted from the CIM, but a complete PIM model cannot be derived just from the CIM.

This way, the transition from high-level business modelling, generally carried out by business analysts, to an executable business process which implies several software functionalities (e.g., web services, components, legacy systems, etc) is far from being a trivial issue [367]. Therefore, the problem of aligning high-level business models (corresponding to the *business view*) and information technologies (corresponding to the *information system view*) became a crucial aspect in the field of software development.

In order to address this issue, we have already started to work on the extraction of valuable information from CIM to PIM models [106]. Besides, we believe that the improvement of transformation engines to support efficient traceabilty mechanisms will help on this task. Therefore, we will keep assessing the performance of model transformation languages regarding traceability support in order to integrate traces management in M2DAT.

6.4.3 Automatic development of Model Transformations, Metamodel Evolution and Model Co-Evolution

The development of model transformations is the most challenging task among those of implementing any proposal for model-driven software development. Besides, constant evolution of metamodels and co-evolution of models has been a common issue to any model-driven proposal [84]. Any modification over a given metamodel (metamodel evolution) implies the need to update conforming models (model co-evolution). Besides, it has another relevant collateral effect: since model transformations are defined at metamodel level, any change over the metamodel has to be subsequently transmitted to any model transformation that use it as soource or target metamodel.

Applying model-driven techniques to support semi-automatic generation of model transformations would help decisively to address these issues.

We have already started to work on two main directions to address this issue. On the one hand, we are carrying out a complete study of existing model transformation engines (part of it has been presented in this thesis) in order to identify the common abstractions used by all of them. The objective is to obtain a common (meta)-metamodel for model transformations. This way, any model transformation could be expressed in terms of such metamodel and translated to any model transformation engine whose metamodel conforms with the afore mentioned.

On the other hand, we have provided with a first case study on the use of *weaving* models and cumulative weaving to automate model migration [356].

6.4.4 Bidirectional Model-to-Text Transformations

When we reviewed existing model transformation languages, we gave a brief overview on existing languages for text-to-model transformations (see section 6.4.6C). This review confirmed that, though there are quite a lot of tools for model-to-text transformations (indeed, any code generator can do it), there are very few works focused on the reverse process and the most of them are focused on the generation of textual syntaxes for DSLs.

Note that one of the advantages of MDE is supposed to be the ability to help on platform migration and interoperability and the first step in such processes implies always the extraction of models from the legacy code, in other words, a text-to-model transformation is needed to produce a model from the existing code. Therefore, MDSD tooling should provide with tools to that end.

As well, that overview highlighted that the main drawback of text-to-model tranformation languages or tools is their starting point. From a grammar specification, such tools generate a textual editor and a metamodel capturing the abstract syntax of the DSL (indeed, it captures the abstract syntax of the grammar). That is, the concrete syntax is defined before the abstract syntax. While it might be acceptable when working with an isolated DSL, it is not a good practice when working with interrelated DSLs connected by means of model transformations. Modifying any given metamodel takes you to the already mentioned complex scenary of metamodel evolution. Therefore, you will need to update any conforming model and any model transformation that used the modified metamodel as source or target metamodel.

We aim at integrating model extraction capabilities in M2DAT, but the above-mentioned approach for text-to-model transformations does not apply since M2DAT metamodels are already defined and they should not be modified. In this sense, a feasible solution is to follow a recent approach to generate textual syntaxes for a DSL. The idea is to start from the metamodel and then define the grammar, generate the editors, etc. In addition, a parser is generated that decides how the text is translated into model elements. Hence, both model-to-text and text-to-model transformations for the given metamodel are obtained. Some works have appeared recently in this direction, though they are still quite immature or badly documented (see [158]). We plan to follow advances in this field in order to integrate injection/extraction capabilities on M2DAT as soon as possible.

6.4.5 Improving the Development of Graphical Editors

Next sextions describe the two main directions that have been detected for further work around graphical's editors development.

6.4.5.1 Automatic development of graphical editors for DSLs

As sections 4.3.3 and 5.2.2.3 stated, GMF is used to build the graphical editors integrated in M2DAT. To that purpose, GMF is based on the definition of a graphical model. It specifies the visual elements that will be used to represent each metaclass from the metamodel that defines the abstract syntax of the DSL. Besides, another model is defined to design the tooling while another model connects the elements from the previous three models (metamodel, graphical and tooling models).

However, it is expected that a graphical or tooling definition may work equally well for several domains. For example, the UML class diagram has many counterparts, all of which are strikingly similar in their basic appearance and structure [120].

A simpler development process would be desirable, where not the graphical definition, neither the correspondence with the abstract syntax have to be defined. In this sense, we have already started to work on the provision of tentative graphical and correspondence models derived directly from the metamodel of the DSL (i.e. the abstract syntax definition). This way, a *default* visual editor (based on boxes and arrows) could be generated once the metamodel had been defined.

6.4.5.2 Improving graphical capabilities of M2DAT

The discussion around approaches for the development of graphical editors of section 4.3.3 highlighted that GMF, though efficient, present some drawbacks, mainly related with the look and feel of the diagrams. Besides, that section confirmed that graphical editors developed useing JAVA Graph components provide with more control over the result. This fact is derived from the generative nature of GMF editors. For instance, the screen captures spread over this dissertation serve to confirm that the graphical features of MIDAS-CASE diagrammers were betther than M2DAT's, though they are much less useful.

Here, the direction for future research would be to combine graphical capabilities of MIDAS-CASE with those from M2DAT. To that end, we need to bring MIDAS-CASe models, persisted in row XML files, to the EMF platform. To accomplish this task we will build technical bridges between MIDAS-CASE's grammarware and M2DAT's modelware following the approach applied by other authors in previous works [44, 45]

This way, we aim at combining the graphical capabilities of JAVA Graph components with that from the functionality provided by EMF in terms of interoperability, etc. In addition, the synergy might work in both directions, since M2DAT capabilities for handling models could be used from a MIDAS-CASE look and feel.

6.4.6 Future works on the Context of M2DAT-DB

Finally, section 6.2 affirmed that M2DAT-DB is a complete result itself. As any other research result, it lends some space for improvement and further work. Next sub-section summarizes the main points in this sense.

- **Extending M2DAT-DBN.** M2DAT-DB aims at providing with a complete framework supporting model-driven development of database schemas, so far it support just OR and XML models. However, we plan to add support for the relational model. Besides, we plan to add support for more DB solutions, like SQL Server or MySQL. To that end, the SQL standard will be used as a pivot model to move between products. Thereby, M2DAT-DB will bundle model transformations to bridge each concrete product model with the standard model and the other way round.
- **Application of M2DAT-DB for computer science teaching**. We have already started to work in the use of M2DAT-DB for educational purposes. The main directions for further work on this line would be:
  - **Model-Driven Engineering**. M2DAT-DB is a complete framework that supports all the common tasks related with implementing a MDSD methodological proposal, like model transformation, model validation or code generation. Hence, it will be used to show the students the heart of a MDSD process. They will learn how to develop model transformations, code generators, graphical editors, weaving models, etc.
  - **Logical Models.** M2DAT-DB supports the object-relational model for databases. Therefore, it will serve to introduce the students in the distinction between a pure object-oriented model and a relational model, as well as in the role of the Object-Relational model to bridge them.
  - **Standards VS Implementations**. Finally, the ability to work both with models compliant to the SQL standard (even generating SQL standard

code) and Oracle models will serve to show the difference between a standard and its implementations.

Appendix A: Resumen en Castellano

Este apéndice ofrece un resumen extendido en castellano de la tesis doctoral que se presenta en esta memoria.

En primer lugar se ofrece una panorámica general de las razones históricas que han llevado a la realización de esta tesis con el objetivo de justificar e identificar claramente los problemas de partida que pretendía atacar en el momento de su realización. A continuación se exponen la hipótesis y principales objetivos de esta tesis, para pasar a presentar la metodlogía seguida durante su desarrollo y concluir cons sus principales aportaciones.

A.1 Antecedentes

A finales del 2000, una nueva forma de concebir el desarrollo de software resultó en un gran grupo de siglas (MDE, MDSD, MDD, DSL, MIC, etc.) que, en realidad, no eran sino distintas formas de referirse a formas de desarrollar software siguiendo una misma aproximación: potenciar el papel de los modelos y las actividades de modelado en cualquier etapa del desarrollo de software. Así, la principal característica del nuevo paradigma de desarrollo, la Ingeniería Dirigida por Modelos (IDM) pasa por centrarse en los modelos en lugar de en los programas [41]. De hecho, la IDM es un paso natural en la tendencia histórica hacia elevar el nivel de abstracción en el desarrollo de software. Cuando aparecieron, los lenguajes de ensamblador, la programación estructurada o los lenguajes orientados a objetos fueron pasos en la misma dirección.

Aunque los modelos habian sido utilizados tradicionalmente en el desarrollo de software, hasta ahora habían desempeñado un papel eminentemente documentativo y, en el mejor de los casos, podían llegar a utilizarse como entrada para la generación de un esqueleto del código final (la herramienta Rational Rose es el ejemplo perfecto de esta tendencia [173]). De este modo, los modelos eran deshechados en cuanto se llegaba a la etapa de codificación y nunca se actualizaban para reflejar los cambios realizados sobre el sistema.

Con la llegada de la IDM el panorama cambia drásticamente, ya que los desarrolladores desplazan su atención del código a los modelos. Así, surge la necesidad de definir modelos lo más precisos y completos posibles, que sean capaces de especificar el sistema a desarrollar y capturar todos sus requisitos, combinándolos con los detalles la plataforma sobre la que se desplegará. Para ello, se parte de modelos de alto nivel de abstracción, que proporcionan detalladas especificaciones del sistema ovbiando detalles tecnológicos. Dichos modelos van siendo refinados hasta alcanzar modelos de bajo nivel que puedan ser directamente traducidos a código fuente.

En realidad, la idea no termina de ser realemente nueva, lo que en realidad confiere un carácter más novedoso a la propuesta es la relevancia que adquiere la automatización del proceso de desarrollo. De hecho, la única forma de hacer realidad las promesas de la IDM en términos de desarrollos más rápidos y baratos pasa por automatizar al máximo el proceso de desarrollo [21, 134]. Como consecuencia, en los últimos años han aparecido numerosas **herramientas para soportar las tareas relacionadas con la IDM** para automatizar cada una de las tareas que implica poner en práctica un proceso de Desarrollo de Software Dirigido por Modelos (DSDM). Así, se encuentran herramientas para definir y utilizar nuevos lenguajes de modelado; herramientas o lenguajes para desarrollar transformaciones de modelos; herramientas para asegurar que los modelos son consistentes y correctos, etc. Cada una de estas herramientas soporta una tarea concreta, es decir, proporciona sólo una parte de la funcionalidad que se necesita para implementar un proceso completo de DSDM. Por ejemplo, el lenguaje de transformación de modelos ATL [184], el más aceptado hasta la fecha, sería una de estas herramientas. Aunque resulta muy potente para su cometido, no es suficiente para desplegar un proceso de desarrollo completo. En el mejor de los casos, se necesitaría de otro lenguaje o herramienta para definir los modelos que ATL se encargará de transformar.

Por otro lado, el impacto de la IDM ha dado lugar a la aparición de propuestas metodológicas de DSDM. Dichas metodologías se basan en la definición y uso de nuevos lenguajes de modelado (bien de propósito general o de propósito específico) para modelar y capturar, a distintos niveles de abstracción, las diferentes partes del sistema a desarrollar. Como consecuencia, apareció un nuevo grupo de herramientas cuyo objetivo era dar soporte a estas propuestas. Así, **las herramientas de soporte a metodologías de DSDM** son entornos de desarrollo para trabajar con el conjunto de modelos interrelacionados que la metodología correspondiente define como necesarios para poder generar el código final que implementa el sistema software. A modo de ejemplo podemos citar ArgoUWE [195], la herramienta que soporta la metodología UWE [198], como una de las herramientas más conocidas en esta categoría.

Los esfuerzos que los autores de estas metodologías dedicaron a construir el soporte técnico para automatizarlas resultó en una serie de herramientas aisladas, que proporcionaban soluciones ad-hoc. Su naturaleza cerrada y la total ausencia de estándares cuando comenzaron a desarrollarse impidieron que se beneficiasen de los avances tecnológicos y la funcionalidad que proporcionan las herramientas para soportar tareas de IDM. Por ejemplo, en ausencia de lenguajes o herramientas para el desarrollo de transformaciones de modelos, los autores optaron por embeber las transformaciones en el código de la propia herramienta, lo que iba claramente en contra de los principios de abstracción y modularidad que rigen el desarrollo de software.

Por todo ello, existe la necesidad de construir nuevas herramientas de soporte para metodologías de DSDM que integren la funcionalidad aislada que propocionan las herramientas existentes para tareas de IDM. Es decir, se deben utilizar las herramientas de soporte a tareas de IDM para construir entornos integrados que implementen metodologías de DSDM.

Cómo construir dicho entorno? En primer lugar, definiendo una arquitectura conceptual que se abstraiga por completo de los detalles técnicos. A continuación, plasmando dicha arquitectura en un diseño técnico que identifique los componentes tecnoloógicos a utilizar. Y finalmente, proporcionando una implementación de referencia de dicho diseño técnico, para demostrar que es viable y factible construir un entorno de desarrollo siguiendo la especificación y cómo debe hacerse.

En este sentido, la pujanza del paradigma de la IDM ha resultado en una tendencia clara hacia la construcción de este tipo de entornos integrados para dar soporte a metodologías de DSDM. No obstante, tal y como muestra el Capítulo 2, no existían este tipo de herramientas cuando abordamos la realización de esta tesis. De hecho, las herramientas de soporte a metodologías de DSDM que adolecían de los problemas comentados, están evolucionando hacia entornos integrados y extensibles, como el que se presenta en esta tesis.

Además, el carácter novedoso de la IDM obliga a hacer especial hincapié en algunos aspectos tradicionalmente relacionados con el desarrollo de herramientas de soporte para tareas de Ingeniería del Software. Tanto la extensibilidad como la inteoperabilidad y la posibilidad de personalizar el entorno son más relevantes si caben cuando hablamos de construir el soporte para una metodología de DSDM. De este modo, la herramienta deberá ser facilmente **extensible** para responder con rapidez a la aparición de nuevos avances en el campo. Por ejemplo, aunque la definición de la semántica de un lenguaje de modelado o las especificaciones formales cobran cada día mayor aceptación como una forma de soportar la simulación y ejecución de modelos [319], el soporte tecnológico para estas tareas se encuentra todavía en fases muy iniciales. No obstante, cualquier herramienta de soporte a una metodología de DSDM debe estar en condiciones de ser extendida para integrar con facilidad el soporte para las tareas mencionadas, en cuanto éste alcance una estabilidad y grado de madurez aceptables.

En el peor de los casos, si la herramienta no soporta una funcionalidad concreta, pero existen otras herramientas que si lo hacen, la herramienta que dará soporte a la metodología debería ser capaz de integrar dicha funcionalidad de forma sencilla. Para ello, los modelos elaborados con la nueva herramienta deberían poder ser facilmente exportados/importados a/desde la herramienta que proporcione la funcionalidad deseada. Por lo tanto, la **interoperabilidad** se convierte en otro de los requisitos clave para herramientas de soporte a metodologías de DSDM.

Igualmente, aunque el objetivo de partida pasa por ser capaces de automatizar el proceso de desarrollo completo propuesto por la metodología, resultaría muy conveniente soportar un proceso de desarrollo que admitiera ciertos puntos de variabilidad, de forma que el diseñador/desarrollador pudiera introducir ciertas decisiones de diseño que dirigieran el resultado final [246]. Por lo tanto, se necesita una forma de **introducir dichas decisiones de diseño** en el proceso de desarrollo, sin reducir el nivel de automatización. Aparte de las decisiones de diseño que ya se hayan recogido en los diferentes modelos, el único modo de introducir decisiones de diseño en el proceso de desarrollo es **soportar transformaciones de modelos *personalizables***.

En este contexto, la tesis que se presenta aborda *la especificación de un entorno para el desarrollo semi-automático de Sistemas de Información Web dirigido por modelos*. Para ello, esta tesis presenta M2DAT *(MIDAS MDA Tool)*, una herramienta para el DSDM que sigue las propuestas metodológicas de MIDAS, una metodología dirigida por modelos para el desarrollo de Sistemas de Información Web (SIW).

Como parte de la propuesta, se define una arquitectura conceptual para la construcción de entornos de DSDM. Dicha arquitectura es modular y dinámica, para facilitar la integración de nuevas funcionalidades en forma de nuevos módulos o subsistemas y soportará la introducción de decisiones de diseño que dirijan la ejecución de las transformaciones de modelos embebidas en la herramienta. Igualmente, se define una aproximación sistemática para la construcción de nuevos módulos de acuerdo a la especificación realizada.

Así mismo, la arquitectura conceptual propuesta será plasmada en un diseño técnico. Esta tarea implica una serie de decisiones de diseño a cerca de cuál es la mejor aproximación y la mejor tecnología para cada tarea relacionada con la

IDM, y cómo debe utilizarse. Así, se realizan y justifican una serie de decisiones tanto metodológicas como tecnológicas, como cuál es la mejor herramienta para definir nuevos lenguajes de modelado; cuál es la mejor aproximación para desarrollar transformaciones de modelos; el mejor lenguaje de entre aquellos que sigan dicha aproximación, etc. Estas decisiones se basan en una completa revisión de la tecnología existente, de acuerdo a una serie de criterios definidos según los requisitos impuestos para la construcción de M2DAT (extensibilidad, interoperabilidad, soporte a transformaciones personalizables, etc.). Como resultado se obtiene una selección de tecnología que identifica la aproximación a seguir para cada tarea, el componente tecnológico a utilizar y las decisiones de diseño que guían el paso de la arquitectura conceptual al diseño técnico de la herramienta.

Finalmente, como parte de esta tesis, se proporciona una implementación de referencia para demostrar que la propuesta es factible, que puede ser utilizada en la práctica y cómo debe hacerse [95]. En particular, se desarrolla M2DAT-DB (MIDAS MDA Tool for DataBases), uno de los módulos de M2DAT. Dicho módulo soporta el desarrollo dirigido por modelos de esquemas de BD modernas. La construcción de M2DAT-DB permite mostrar que, tanto la especificación conceptual como el diseño técnico propuestos, así como las decisiones metodológicas y técnicas que permiten el paso de uno al otro, y el proceso de desarrollo propuesto para la construcción de nuevos módulos, son apropiados para implementar propuestas metodológicas de DSDM.

A.2 Objetivos

A continuación se exponen la hipótesis y principales objetivos de esta tesis.

La **hipótesis** formulada en esta tésis es que "*es factible proporcionar una solución técnica para la construcción de un entorno que soporte el desarrollo semi-automático dirigido por modelos de Sistemas de información Web, utilizando las herramientas y componentes existentes a día de hoy en el contexto de la IDM*"

Por lo tanto, el **objetivo principal** de esta tesis, derivado directamente de la hipótesis, es *"proporcionar una solución técnica para la construcción de un entorno que soporte el desarrollo semi-automático dirigido por modelos de Sistemas de información Web, utilizando las herramientas y componentes existentes a día de hoy en el contexto de la IDM*"

Este objetivo se desglosa en una serie de objetivos parciales:

O1. Análisis y evaluación de la tecnologñia existente (herramientas de soporte para tareas de IDM) de cara a identificar las más apropiadas para construir un entorno para soportar el desarrollo dirigido por modelos de SIWs. De acuerdo a las tareas concretas que implica la construcción de dicho entorno, podemos descomponer este objetivo como sigue:

- **O1.1.** Análisis y evaluación de herramientas de (meta)modelado.
- **O1.2.** Análisis y evaluación de motores de transformación de modelo-a-modelo, haciendo especial hincapié en el soporte para la introducción de decisiones de diseño.
- **O1.3.** Análisis y evluación de motores de transformación modelo-a-texto (también referidos como generadores de código).
- **O1.4.** Análisis y evaluación de herramientas de soporte para el resto de tareas relacionadas con la IDM, como desarrollo de editores gráficos o validadores de modelos..

O2. Análisis y evaluación de entornos que soporten propuestas de DSDM.

- **O2.1.** Analisis y evaluación de entornos para el desarrollo dirigido por modelos de SIWs..
- **O2.2.** Análisis y evaluación de entornos para el desarrollo dirigido por modelos de esquemas de BD modernas (objeto-relacionales y XML).

O3. Especificación de la arquitectura conceptual de M2DAT.

O4. Selección de tecnologías a emplear para construir M2DAT.

O5. Especificación del diseño técnico de M2DAT.

O6. Especificación del proceso de desarrollo para cada módulo de M2DAT.

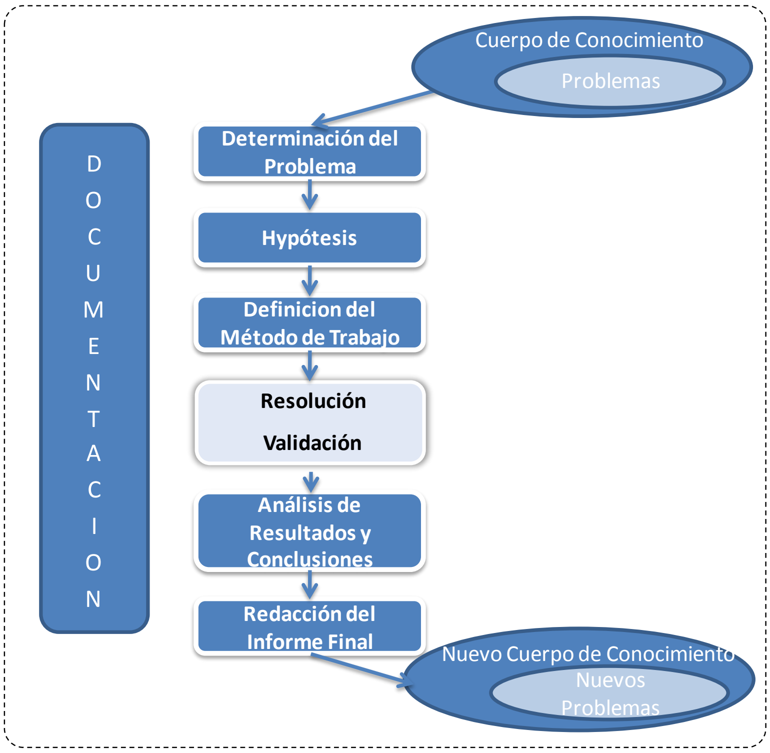
O7. Validación del diseño técnico de M2DAT. Para ello, se plantean dos sub-objetivos:

- **O7.1.** Construcción de M2DAT-DB, uno de los módulos de M2DAT, que actúa a modo de **prueba de concepto** para la propuesta (arquitectura conceptual, diseño técnico, selección de tecnología y proceso de desarrollo de nuevos módulos).
- **O7.2.** Desarrollo de casos de estudio con M2DAT-DB.

A.3 Metodología

La diferente naturaleza de las Ingenierías respecto al resto de ciencias empíricas y formales imposibilitan la aplicación directa de métodos clásicos a la investigación en Ingeniería del Software. Así, el método de investigación que se sigue en esta tesis está adaptado del propuesto en [223] para la investigación en ingeniería del Software. Se basa en el método hipotético–deductivo de Bunge [67], y se compone de varias etapas que, dada su genericidad, son aplicables a cualquier tipo de investigación.

Tal y como muestra la Figura A-1, la definición del método de investigación es un paso del propio método. Dicho pasa es necesario porque cada proceso de investigación posee sus propias características. Por lo tanto, no hay un método universal que pueda aplicarse a cualquier trabajo de investigación.


Cuerpo de Conocimiento
Problemas
DOCUMENTACION
Determinación del Problema
Hypótesis
Definicion del Método de Trabajo
Resolución
Validación
Análisis de Resultados y Conclusiones
Redacción del Informe Final
Nuevo Cuerpo de Conocimiento
Nuevos Problemas
Figura A-1. Método de Investigación

Dado que la fase más importante de dicho método es la de resolución y validación, a continuación se proporciona una vista más amplia del proceso seguido en esta fase, que en cierto modo es una adaptación del tradicional proceso en cáscada [300] y el Proceso Unificado de Rational [177]. La Figura A-2 muestra una vista simplificada del proceso.

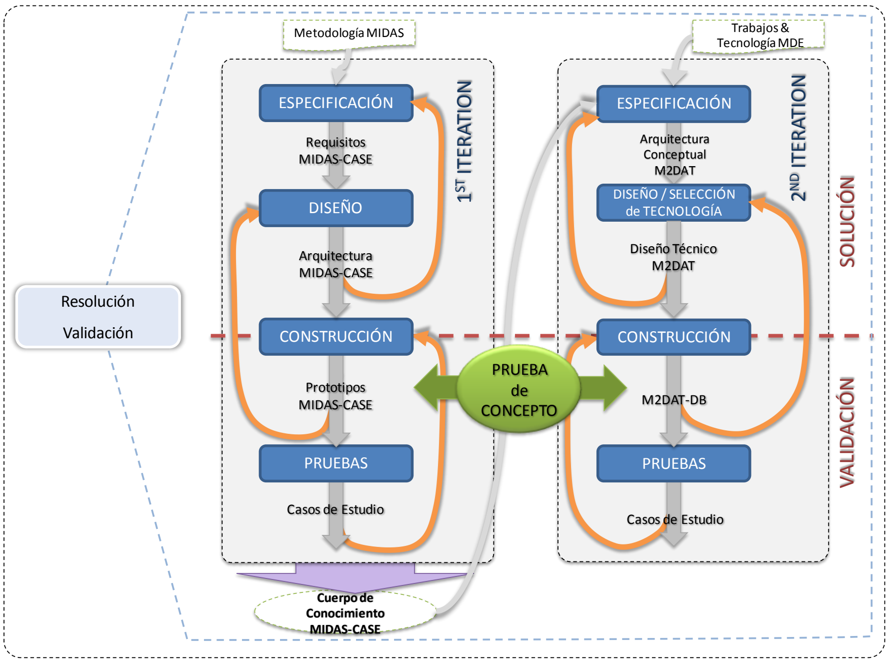


Figura A-2. Fase de Resolución y Validación del Método de Investigación

Primera Iteración: desarrollo de MIDAS-CASE

Durante la fase de **especificación** de la primera iteración se revisan los trabajos relacionados con herramientas CASE y la metodología MIDAS. El objetivo es identificar las necesidades relacionadas con dar soporte a MIDAS y si las herramientas existentes podían satisfacerlas. Dicha revisión conluye con la decisión de construir un nuevo entorno para soportar la representación gráfica de los modelos propuestos en MIDAS, el paso de unos a otros y la generación de código a partir de dichos modelos. Además, se establece el uso de una BD XML como repositorio de modelos. Igualmente se identifican los dos requisitos más importantes que dicho entorno debía reunir: extensibilidad y modularidad.

La fase de **diseño** se relaciona fundamentalmente con la definición de la arquitectura del nuevo entorno, de acuerdo a los requisitos establecidos durante la fase de especificación. Además, se identifican los componentes tecnológicos a utilizar y el proceso de desarrollo a seguir para construir cada módulo. La principal salida de esta fase es la arquitectura de MIDAS-CASE, que combina la arquitectura conceptual con el diseño técnico.

Para validar los resultados de la fase de diseño, se construyen dos prototipos durante la fase de **construcción**: MIDAS-CASE4WS y MIDAS-CASE4XS, que soportan, respectivamente, el modelado de Servicios Web y XML Schemas con UML extendido y la serialización de dichos modelos en código final (WSDL y XSD). Estos prototipos son la **prueba de concepto** para la especificación de la arquitectura de MIDAS-CASE.

Finalmente, la fase de **pruebas** consiste en el desarrollo de una batería de casos de estudio con los prototipos de MIDAS-CASE para evaluar la viabilidad y utilidad de la propuesta, así como para mejorar la arquitectura y el proceso de desarrollo de nuevos módulos definidos durante la fase de diseño.

Nótese que cada paso del proceso realimenta los anteriores. Por ejemplo, los hallazgos y lecciones aprendidas obtenidos en la fase de construcción de prototipos influyen en la fase de diseño de cara a refinar la arquitectura de la herramienta.

Segunda Iteración: desarrollo de M2DAT

Tras concluir con el desarrollo de MIDAS-CASE se comienza una nueva iteración con dos objetivos principales: considerar, estudiar e incorporar los avances en el campo de la IDM y aprovechar las lecciones aprendidas durante la primera iteración para solventar o paliar los principales problemas y deficiencias encontrados en las herramientas que soportan propuestas metodológicas para el DSDM.

Por lo tanto, durante la fase de **especificación** se realiza una revisión de la tecnología existente para dar soporte a las tareas relacionadas con la IDM y de las lecciones obtenidas del desarrollo de MIDAS-CASE. Una conclusión importante pasa por separar la definición de la arquitectura del entorno de desarrollo, a alto nivel, de su descripción técnica, a bajo nivel. Así, la salida principal de esta fase es la arquitectura conceptual de M2DAT, la nueva versión de la herramienta de soporte para MIDAS, y los conocimientos técnicos necesarios para abordar la fase de diseño.

Durante la fase de **diseño**, la arquitectura conceptual se refina en un diseño técnico de acuerdo al conocimiento adquirido de las revisiones de tecnología realizadas durante la fase de especificación. Frente a la arquitectura de MIDAS-CASE, M2DAT se define a partir de varias herramientas que soportan tareas concretas relacionadas con la IDM sobre la plataforma Eclipse y más concretamente, el *Eclipse Modelling Framework* (EMF) [66, 161]. El resultado es un entorno muy fácilmente extensible, que puede integrar nuevas funcionalidades a medida que sean liberadas. Así mismo, se define el proceso de desarrollo para nuevos módulos de acuerdo al diseño técnico de M2DAT.

Dicho diseño guía la **construcción** de M2DAT-DB, la **prueba de concepto** para M2DAT, durante la fase de construcción. M2DAT-DB sirve como implementación de referencia para probar y validar la viabilidad del diseño técnico de M2DAT y del proceso de desarrollo de nuevos módulos. Al mismo tiempo, M2DAT-DB sirve de guía sobre cómo seguir la especificación de M2DAT a la hora de construir e integrar nuevos módulos en la herramienta.

Finalmente, una batería de casos de estudio llevados a cabo con M2DAT-DB durante la fase de pruebas ayudan en el refinamiento de la propuesta. Por ejemplo, la necesidad de disponer de transformaciones de modelos personalizables se detectó durante el desarrollo de estos casos de estudio. Concecuentemente, el diseño técnico de M2DAT y el proceso de desarrollo de nuevos módulos (en particular de las transformaciones de modelos) fueron modificados de acuerdo a la nueva necesidad.

A.4 Conclusiones

Esta tesis proporciona una serie de contribuciones, no sólo en el ámbito de la investigación planteada en el punto de partida (la especificación de M2DAT), sino también relacionadas con otros aspectos colaterales. Se resumen a continuación.

Un completo análisis de las soluciones existentes para la construcción de entornos que soporten metodologías para el desarrollo dirigido por modelos de Sistemas de Información Web.

Todas las decisiones técnicas que se recogen en la especificación de M2DAT han sido explicadas y justificadas a lo largo de esta tesis. No obstante, en el camino hacia esas decisiones se ha realizado una completa revisión de la tecnología existente en el campo de la IDM. Dicha revisión constituye en sí misma una contribución relevante, dado que podría utilizarse para otros propósitos más concretos.

De hecho, algunos de los entornos para soportar metodlogías de DSDM revisados están siendo mejorados (o al menos dicha mejora se ha planificado) para adaptarlos a los avances en el campo. Los hallazgos y el análisis proporcionados en el estado del arte de esta tesis podrían ayudar a los investigadores responsables de esos trabajos a la hora de realizar la selección de tecnología que mejor se adapte a las necesidades particulares de su metodología, que no tienen por qué ser exactamente las mismas que plantea M2DAT.

Por ejemplo, la forma en que dichos entornos soportan las transformaciones de modelos es un caspecto susceptible de mejora en todos los casos estudiados. En este sentido, esta tesis proporciona una serie de conclusiones y lecciones aprendidas que ayudarán a incoporar las ventajas de la tecnología actual a dichos entornos, como el uso de modelos de anotación o las ventajas de usar una aproximación híbrida frente a una puramente declarativa o imperativa.

Especificación de un entorno para el desarrollo dirigido por modelos de Sistemas de Información Web

La principal contribución de esta tesis ha sido la especificación de M2DAT, un entorno abierto que soporta el desarrollo dirigido por modelos de Sistemas de Información Web. Las principales aportaciones de M2DAT con respecto a los trabajos existentes pasan por su naturaleza **extensible** e **interoperable** y el soporte de transformaciones personalizables mediante modelos de anotación.

Estas características contribuyen a facilitar la tarea de extender la herramienta para que soporte nuevas funcionalidades. Siempre que MIDAS, la metodología soportada por M2DAT, sea extendida para incluir un nuevo aspecto en la arquitectura de MIDAS, se desarrollará un nuevo módulo de M2DAT que soportará la extensión de la metodología. Además, los modelos que el nuevo módulo soporte serán interoperables con los ya soportados sin necesidad de realizar ningún esfuerzo adicional. Es decir, no será preciso construir puentes entre espacios técnológicos [208], como sucedía hasta la fecha. De hecho, aunque a priori todas las herramientas de modelado podrían considerarse en el mismo espacio tecnológico, la práctica demuestra que esta afirmación no es del todo cierta. Importar y exportar modelos desde una herramienta a otra implica moverse entre distintos espacios tecnológicos, en general el de las gramáticas y el de los modelos (*grammarware* y *modelware*). Esta tarea, a pesar de que existen soluciones técnicas que ayudan a llevarla a cabo, suele implicar perdida de semántica y resulta muy propensa a introducir errores. En cambio, cuando los modelos se definen a partir de un metametamodelo común, el único artefacto necesario para conectarlos es una transformación de modelos, y, opcionalmente, una modelo de *weaving* para establecer la forma en qué llevarla a cabo.

Por otro lado, ninguno de los trabajos existentes ofrecía soporte para **transformaciones personalizables**. De esta manera, cuando el proceso de desarrollo propuesto por la metodología era completamente automatizado, la única forma de mantener algún control sobre el resulatdo final era modificar los modelos de entrada. Incluso después de modificarlos, algunas de las transformaciones incluidas en dichas herramientas no eran capaces de producir ciertas construcciones en los modelos de salida. La úncia forma de hacerlo era modificar manualmente los modelos de salida.

Frente a este comportamiento, las transformaciones de modelos incluidas en M2DAT utilizan modelos de anotación para controlar la ejecución de la transformación. Así, se pueden obtener diferentes modelos de salida a partir de un mismo modelo de entrada sin más que modificar el modelo que contiene las anotaciones. De esta forma se soporta la personalización del proceso de desarrollo sin que ello perjudique el nivel de automatización. Además, en ausencia de anotaciones, dichas transformaciones incorporan un comportamiento por defecto. Una ventaja adicional es que, de esta manera, las decisiones de diseño son guardadas y pueden ser recuperadas, consultadas y modificadas en cualquier momento. Por último, el artefacto para guardar dichas decisiones es el más adecuado en el contexto de la IDM: un modelo.

Finalmente, la especificación de M2DAT tambien comprende la definición del proceso de desarrollo a seguir para construir nuevos módulos, así como la implementación de referencia, M2DAT-DB, que constata que dicha especificación es viable y ayuda a interpretarla.

Soporte para el desarrollo semi-automático dirigido por modelos de esquemas de BD modernos.

Aunque el objetivo de desarrollar M2DAT-DB era fundamentalmente proporcionar una prueba de concepto para la propuesta de M2DAT, M2DAT-DB constituye en si mismo una herramienta completa para el desarrollo dirigido por modelos de esquemas de BD modernas. Dado que el estado del arte ha servido para constatar que a día de hoy no existen herramientas que soporten toda la funcionalidad proporcionada por M2DAT-DB, podemos decir que ésta es en sí misma una aportación relevante de la tesis.

M2DAT-DB proporciona un conjunto de herramientas para trabajar con DSLs para el modelado de esquemas de BD Objeto-Relacionales, tanto para el estándar SQL:2003, como para el producto comercial Oracle. Dichas herramientas permiten generar el esquema de la BD a partir de un modelo conceptual de datos representado con un diagrama de clases UML. Además, el proceso de generación puede ser personalizado mediante la definición de un modelo de anotación que recoja las decisiones de diseño que especifican cómo se quiere mapear un elemento concreto del modelo conceptual. Igualmente, la herramienta incluye las transformaciones para pasar de un modelo Obejto-Relacional conforme al estándar a uno para Oracle y viceversa, así como editores gráficos para manejar los modelos elaborados.

Finalmente, para completar el soporte al aspecto del contenido, M2DAT-DB incluye también las facilidades para el desarrollo de esquemas XML siguiendo la misma aproximación. Es decir, el modelo XML se obtiene automáticamente a partir del modelo conceptual de datos y el proceso de generación deja espacio para la personalización mediante modelos de anotación.

Appendix B: About Graph Transformations

This sections aims at putting forward some commonalites on graph transformations in order to help on the understanding of the reviewed works that provide with a graph-based transformation language.

B.1 Graph-Based Model Transformation Languages

Graph transformation rules consist of a LHS graph pattern and a RHS graph pattern. When applying a graph transformation rule, every match with the left-hand-side graph pattern is replaced by the right-hand-side graph pattern. In addition to the left-hand-side graph pattern, non-matching conditions can be defined, e.g. negative conditions.

The graph patterns can be rendered in the concrete syntax of their respective source or target language (e.g., in VIATRA) or in the MOF abstract syntax. The latter is the way it is done in AGG and BOTL.

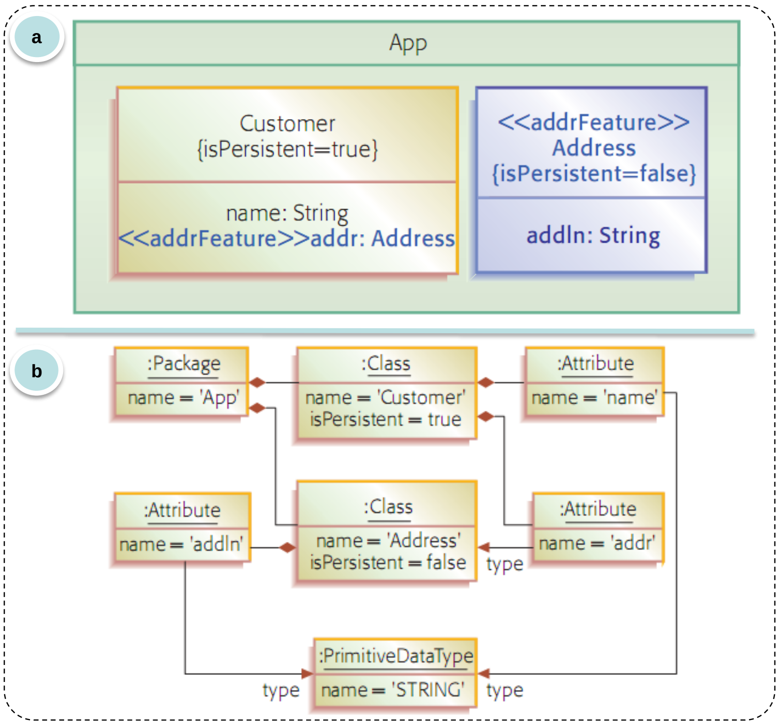


Figure B-1. UML simple model rendered in UML concrete syntax (a) and in MOF abstract syntax (b)

As Figure B-1 shows (retrieved from [98]), models expressed in the concrete syntax are more familiar to developers working with a given modelling language. Besides, for complex languages like UML, patterns in a concrete syntax tend to be much more concise than patterns in the corresponding abstract syntax.

The kernel of some graph-based languages is defining triple graph grammars (TGG), a particular type of graph-based transformation approach introduced by Andy Schurr [313]. In addition to the graphs used to specify the LHS and RHS of the rule, Triple Graph Grammar rules use a third sub-graph, called correspondence graph. Its elements are linked to the source and target elements located at the LHS and RHS. In essence, the correspondence graph holds the tracing information of the transformation. A very simple example, taken from [388], is shown in Figure B-2.

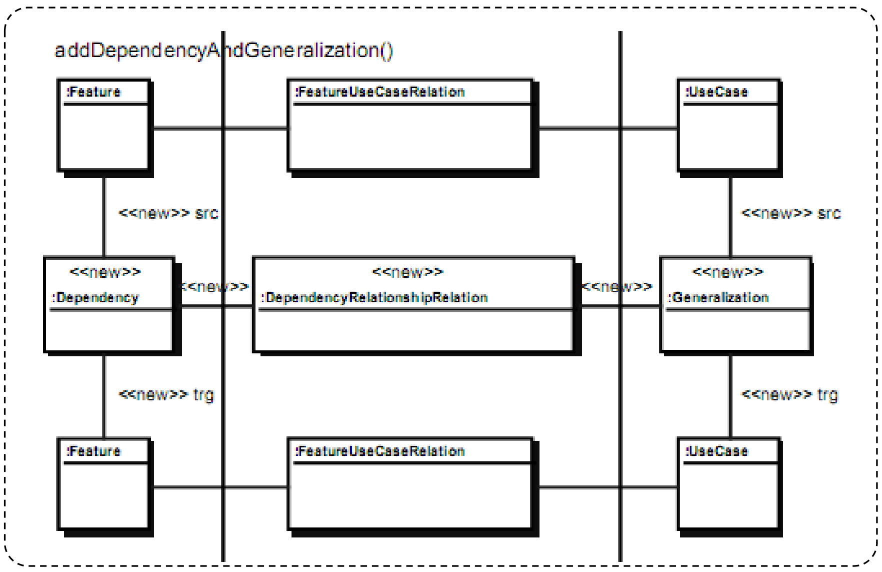


Figure B-2. Example of Triple Graph Grammar (TGG) rewriting rule

Besides, it is also very common to provide support for definition of Negative Application Condition (NAC). NACs specify conditions that should not be present in the host graph in order for the rule to be applied. Figure B-3, adapted from [382], uses a simplified example to show the use of NAC rules. It is based on the Pac Man game, commonly used in graph grammars literature. Pac Man can move in two ways. It might go to a field that contains a marble. Then, the marble disappears, Pac Man is located in the next field and increments his count of marbles (Figure B-3 (a)). On the other hand, Pac Man next movement could be to a field with no marble in. This is a NAC, and it is represented as a crossed out object (Figure B-3 (b)).

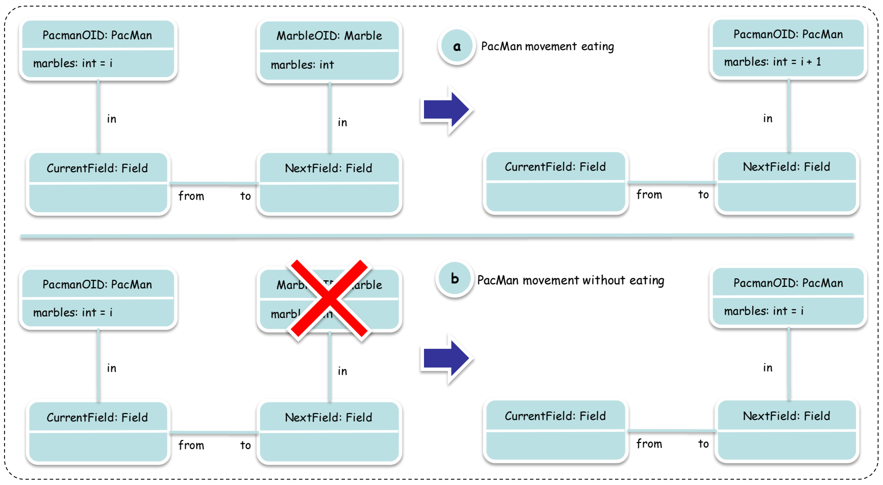


Figure B-3. Production rules for PACMAN game: (a) Pac Man movement eating (b) Pac Man movement without eating

Finally, although many graph-based model transformation languages use a visual notation for specifying the transformation, identifying graph-based transformations with visual notations is a common mistake. In fact, many of them support (just) textual notations, like VIATRA.

Appendix C: Text-to-Model Transformations

This section has been added at the time of finishing this dissertation in order to cope with an emerging research field in the context of MDE, closely related with the development of tools for MDSD: text to model transformations. Since we plan to integrate support for this task in M2DAT, we have decided to include a review of existing works in the field. This section summarizes our main findings.

C.1 Text-to-Model Transformation approaches

Obtaining models from code is an area of increasing interest is. Indeed, the Architecture Driven Modernization (ADM, http://adm.omg.org/) initiative from the OMG is related to extending the modelling approach to the existing software systems. It is a kind of reverse engineering whose first step should be producing models from legacy code, also known as model injection, the opposite of code generation (extraction). In contrast with model-to-text transformation languages, the underlying objective of text-to-model transformation languages is the mapping of concrete syntaxes to abstract syntaxes. To that purpose, these languages might follow a **grammar-based** or a **metamodel-based** approach.

The former are focused on the definition of a context-free grammar [83] to specify the concrete syntax of the language. The metamodel that defines the abstract syntax is derived from the grammar, together with a textual editor (a parser indeed) for conforming models. This approach have been traditionally adopted to build compilers for programming languages and is currently adopted by the Xtext [146] component of OpenArchitectureWare (see section 2.2.13).

On the other hand, metamodel-based approaches start from the metamodel and provides with a (user-friendly) grammar language to define the grammar for your DSL. Such language for grammars allows specifying a textual representation for each concept of the metamodel. Doing so, you are defining the (textual) concrete syntax of the DSL. From that, the framework generates textual editors for the new DSL and even in some cases, like TEF [312], synchronically updated with the corresponding visual editor

Whereas grammar-based approaches worked fine for compilers development, they fail when it comes to MDE. Their main drawback is the starting point. To specify a text-to-model transformation following a grammar-based approach, you have to, first, define the grammar. Then, the metamodel is automatically derived from the grammar specification. This approach suffers of the inability to create a custom metamodel. While it might be acceptable when working with an isolated DSL, it is not a good practice when working with interrelated DSLs, since the metamodels are to be the ports where model transformation are anchorage to connect the DSLs. In fact, since you are defining the concrete syntax before defining the abstract syntax, the generated metamodel is in essence a high-level representation of the grammar, more than an abstraction of the domain concepts you want your DSL to model.

As Markus Scheidgen argued in his blog [311], MDE changes the software engineering scene and the way IDEs have to be developed. Before, those IDEs were only tools to help on coding tasks. At present, they have to be (meta)modelling frameworks providing with support for all types of model processing tasks, such as analysis, validation, transformation, etc. In this context, textual editors have to be developed in a different way. Existing frameworks ignore that most DSLs contains non-context-free features that are described in the corresponding metamodel. The starting point to develop a textual editor for a given DSL should be the metamodel and not the grammar. From the metamodel, you should specify the grammar, generate graphical and textual editors, etc.

We believe that bidirectional bidirectional mappings that supports both model-to-text (extraction), and text-to-model transformations (injection) [255] are the way to address this task. This approach eases the development of bidirectional editors, supporting both injection (text-to-model) and extraction (model-to-text) of models.

Working this way you get an extra contribution, that is the interesting one for us. Along with the textual editor, injectors/extractors are generated. As far as we know, only TCS [183] is providing (in an minimum reliable manner) with this functionality and has been applied in several projects. Unfortunately, developing TCS specifications is a challenging task and lacking of documentation.

Due to it is still an emerging field, we will not make an exhaustive comparative here on works supporting some kind of text-to-model transformations and we refer the reader to the first works comparing exiting works for this task [158]. However, since we plan to integrate support for models extraction in M2DAT, in the following we present the main features of existing languages to specify model extraction processes and we will follow advances in the field in order to integrate support for text-to-model transformations in M2DAT as soon as possible.

C.2 Languages for bidirectional mapping of text to models

In the following we summarize the main features of the main text-to-model transformation languages supporting bidirectional mapping of model-to-text and text-to-model

TCS

The Textual Concrte Syntax (TCS, [183]) is an Eclipse/GMT component that enables the specification of textual concrete syntaxes for DSLs by attaching syntactic information to metamodels. With TCS, it is possible to parse (text-to-model) and pretty-print (model-to-text) DSL sentences. Moreover, TCS provides an Eclipse editor, which features: syntax highlighting, an outline, hyperlinks, and hovers for every DSL which syntax is represented in TCS.

To that end, TCS provides with a DSL for the specification of the correspondence between the metamodel and its textual representation. From that, an ANTLR grammar together with a parser for this grammar is generated. Such parser (also know as injector) takes as input a textual program of the DSL and generates a model conforming to the DSL metamodel. In addition, TCS also generates an extractor that provides with model-to-text capabilities (code generation in fact).

TCS also has some limitations. Despite technical concerns, we might notice that the mapping is too complex when the metamodel is far from the desired syntax. For example it is currently impossible to create blank delimited or case insensitive languages. Besides, there is not much documentation available and coding TCS mappings is much more complex than ATL transformations.

Xtext

Xtext [124] is a language of the OpenArchitectureWare framework (see section 2.2.13) that enables text-to-model transformations.

In contrast with TCS, that followed the metamodel-based approach, Xtext follows the grammar-based approach. Hence, the metamodel is derived from a Xtext grammar file that describes the syntax of the DSL. From the grammar specification an ANTLR grammar and a Ecore metamodel are generated. The generated metamodel corresponds to an AST specification for the DSL. Besides, Xtext generates an Eclipse-based textual editor for the DSL.

The parser generated works as an injector that creates model elements from textual specifications. Its main issue resides in the approach followed, since the generated metamodel is basically an abstraction of the concrete syntax. To overcome this drawback the authors propose [146] to transform from the generated metamodel into the intended target metamodel. That is, the model transformation would translate a model of the concrete syntax to a model of the abstract syntax.

At present, there is ongoing work to integrate both TCS and Xtext under the Eclipse Textual Modelling Framework project (http://www.eclipse.org/modeling/tmf/).

Sintaks/TCSSL

Sintaks [135, 255] uses bidirectional mapping-models to support both model-to-text and text-to-model transformations (generators and parsers). Thereby, it defines bridges between concrete (textual files) and abstract syntax (models). Additionally, Sintaks allows generating automatically textual editors for a model providing syntax highlighting. Sintaks is based onto the EMF repository.

Appendix D: Case Study

This Chapter presents the Case Study used all along Chapter 5 to show the application of the reference implementation of M2DAT.

This Case Study is taken from [385] (p. 5). As mentioned before, using an “external” case study prevent us from using ad-hoc models that might fit better to our needs. The Online Movie Database (OMDB) is devised to manage information about movies, actors, directors, play writers and movie related information. Users can browse this information on the OMDB website and purchase products (i.e. movie videos, DVDs, books, CDs, and other movie related merchandise). The movie information includes the movie title, director, the official movie website, genre, studio, short synopsis, and the cast (i.e. actors and the roles they play in the movie). Each movie has up to 5 external editorial reviews, and unlimited number of user reviews entered by users online. OMDB website offers products for sale including movie videos and DVDs. Information about videos and DVDs includes title, rating, list price, release date, and other relevant information. This situation can be modelled using the UML Class Diagram shown in Figure D-1.

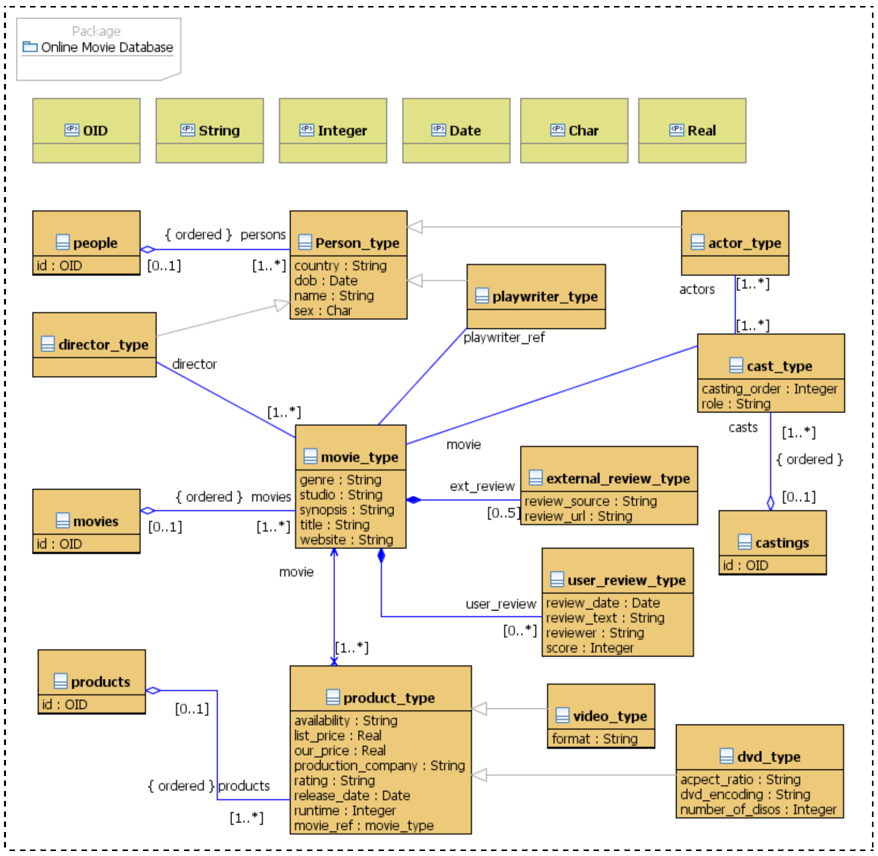
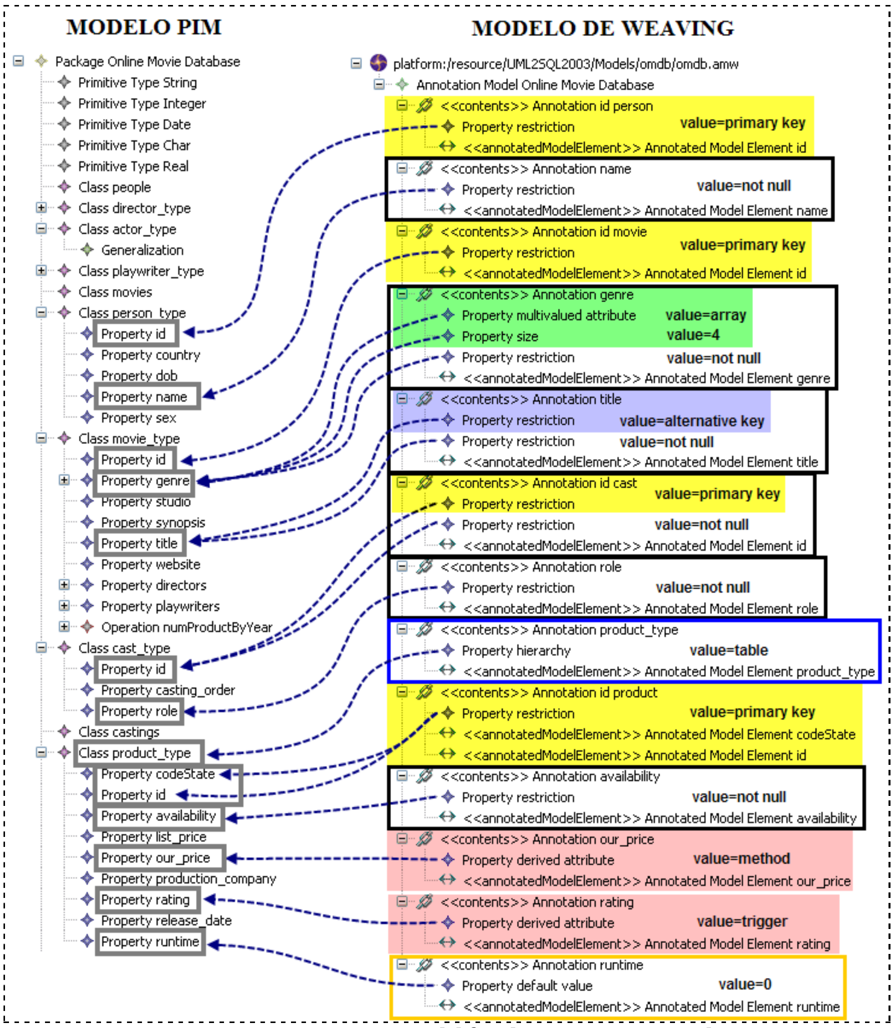


Figure D-1. Conceptual Data Model for the OMDB Case Study

Using M2DAT-DB, such conceptual data model can be translated to an ORDB schema conforming to the SQL:2003 standard, and ORDB Schema for Oracle or an XML Schema. Such mapping could be driven by a set of annotations. Since we will focus on the DSL for modelling ORDB schemas to show model edition and validation capabilities in M2DAT, Figure D-2 shows the conceptual model next to the correspondent annotation model for generating an ORDB model for SQL:2003.



MODELO PIM

```
Package Online Movie Database
  Primitive Type String
  Primitive Type Integer
  Primitive Type Date
  Primitive Type Char
  Primitive Type Real
  Class people
  Class director_type
  Class actor_type
    Generalization
  Class playwriter_type
  Class movies
  Class person_type
    Property id
    Property country
    Property dob
    Property name
    Property sex
  Class movie_type
    Property id
    Property genre
    Property studio
    Property synopsis
    Property title
    Property website
    Property directors
    Property playwriters
    Operation numProductByYear
  Class cast_type
    Property id
    Property casting_order
    Property role
  Class castings
  Class product_type
    Property codeState
    Property id
    Property availability
    Property list_price
    Property our_price
    Property production_company
    Property rating
    Property release_date
    Property runtime
```

MODELO DE WEAVING

```
platform:/resource/UML2SQL2003/Models/omdb/omdb.amw
  Annotation Model Online Movie Database
    <<contents>> Annotation id person
      Property restriction                         value=primary key
      <<annotatedModelElement>> Annotated Model Element id
    <<contents>> Annotation name
      Property restriction                         value=not null
      <<annotatedModelElement>> Annotated Model Element name
    <<contents>> Annotation id movie
      Property restriction                         value=primary key
      <<annotatedModelElement>> Annotated Model Element id
    <<contents>> Annotation genre
      Property multivalued attribute               value=array
      Property size                                value=4
      Property restriction                         value=not null
      <<annotatedModelElement>> Annotated Model Element genre
    <<contents>> Annotation title
      Property restriction                         value=alternative key
      Property restriction                         value=not null
      <<annotatedModelElement>> Annotated Model Element title
    <<contents>> Annotation id cast
      Property restriction                         value=primary key
      Property restriction                         value=not null
      <<annotatedModelElement>> Annotated Model Element id
    <<contents>> Annotation role
      Property restriction                         value=not null
      <<annotatedModelElement>> Annotated Model Element role
    <<contents>> Annotation product_type
      Property hierarchy                           value=table
      <<annotatedModelElement>> Annotated Model Element product_type
    <<contents>> Annotation id product
      Property restriction                         value=primary key
      <<annotatedModelElement>> Annotated Model Element codeState
      <<annotatedModelElement>> Annotated Model Element id
    <<contents>> Annotation availability
      Property restriction                         value=not null
      <<annotatedModelElement>> Annotated Model Element availability
    <<contents>> Annotation our_price
      Property derived attribute                   value=method
      <<annotatedModelElement>> Annotated Model Element our_price
    <<contents>> Annotation rating
      Property derived attribute                   value=trigger
      <<annotatedModelElement>> Annotated Model Element rating
    <<contents>> Annotation runtime
      Property default value                       value=0
      <<annotatedModelElement>> Annotated Model Element runtime
```

Figure D-2. Annotation Model for the OMDB Case Study

Figure D-3 shows the resuling ORDB model for SQL:2003.

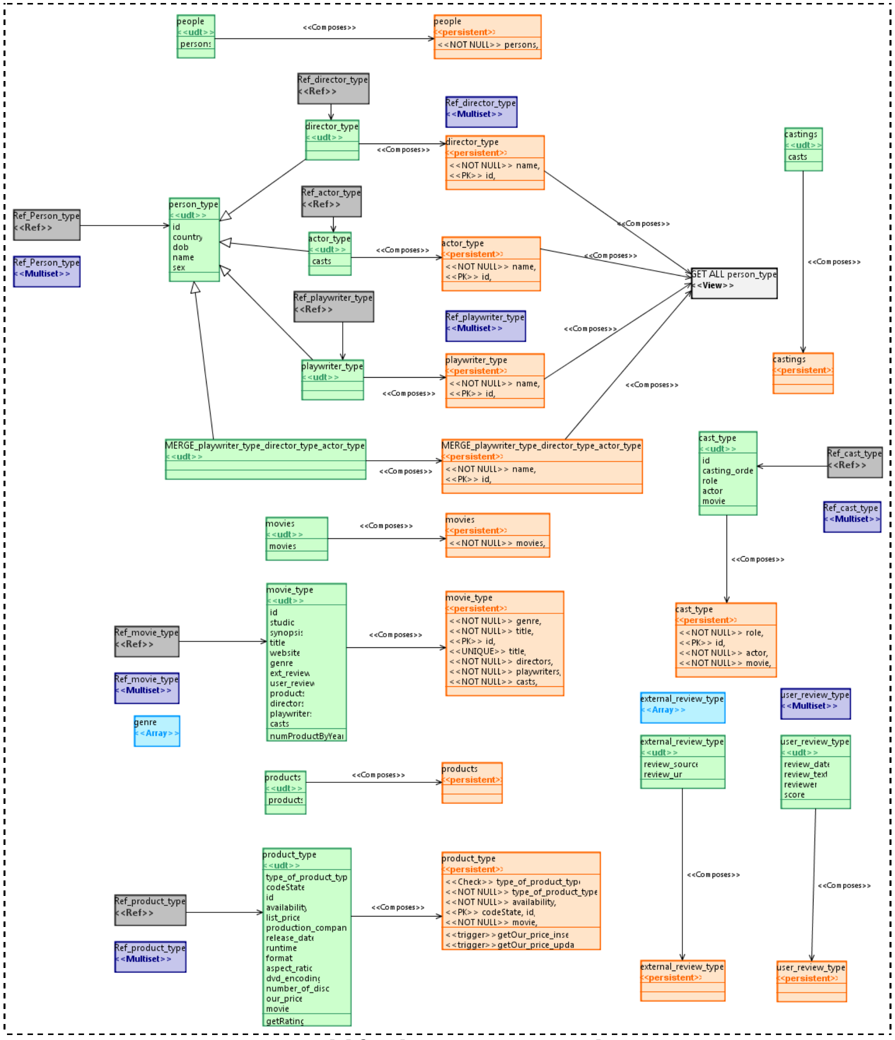
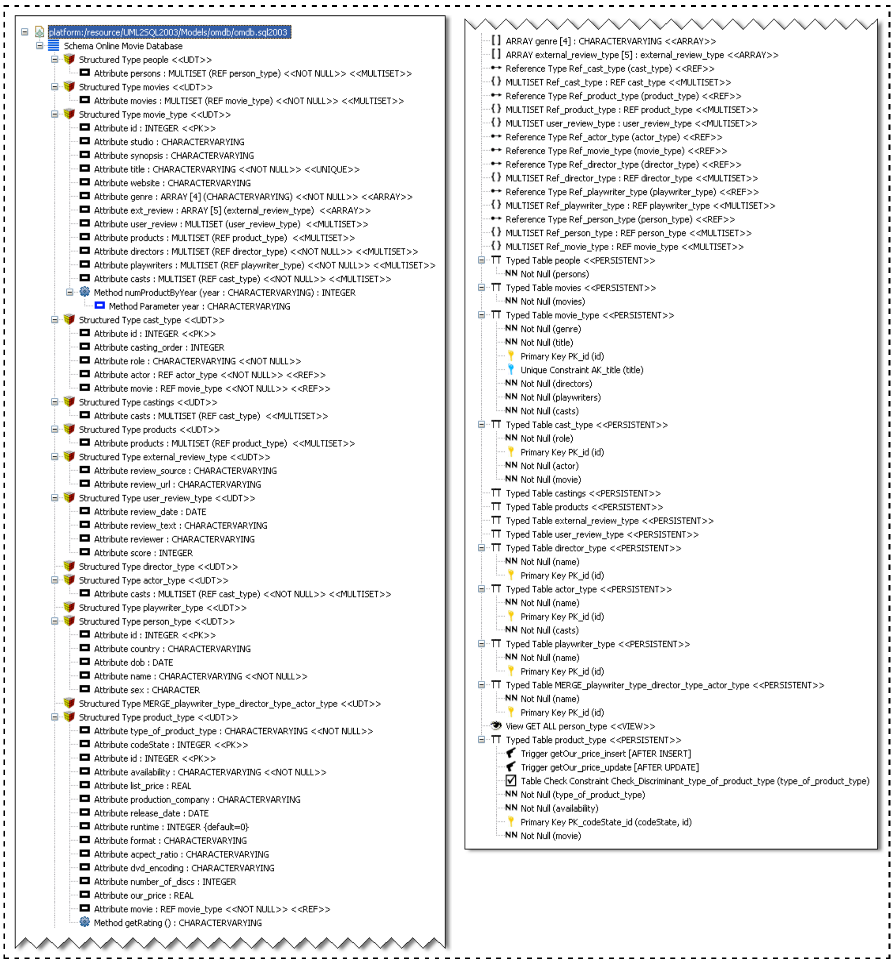

people <<udt>> persons
<<Composes>>
people <<persistent>> <<NOT NULL>> persons,
Ref_director_type <<Ref>>
Ref_director_type <<Multiset>>
director_type <<udt>>
director_type <<persistent>> <<NOT NULL>> name, <<PK>> id,
castings <<udt>> casts
Ref_Person_type <<Ref>>
Ref_Person_type <<Multiset>>
person_type <<udt>> id country dob name sex
Ref_actor_type <<Ref>>
actor_type <<udt>> casts
actor_type <<persistent>> <<NOT NULL>> name, <<PK>> id,
GET ALL person_type <<View>>
Ref_playwriter_type <<Ref>>
Ref_playwriter_type <<Multiset>>
playwriter_type <<udt>>
playwriter_type <<persistent>> <<NOT NULL>> name, <<PK>> id,
castings <<persistent>>
MERGE_playwriter_type_director_type_actor_type <<udt>>
MERGE_playwriter_type_director_type_actor_type <<persistent>> <<NOT NULL>> name, <<PK>> id,
cast_type <<udt>> id casting_orde role actor movie
Ref_cast_type <<Ref>>
Ref_cast_type <<Multiset>>
movies <<udt>> movies
movies <<persistent>> <<NOT NULL>> movies,
movie_type <<udt>> id studic synopsis title website genre ext_review user_review products directors playwriters casts numProductByYear
movie_type <<persistent>> <<NOT NULL>> genre, <<NOT NULL>> title, <<PK>> id, <<UNIQUE>> title, <<NOT NULL>> directors, <<NOT NULL>> playwriters, <<NOT NULL>> casts,
Ref_movie_type <<Ref>>
Ref_movie_type <<Multiset>>
genre <<Array>>
cast_type <<persistent>> <<NOT NULL>> role, <<PK>> id, <<NOT NULL>> actor, <<NOT NULL>> movie,
external_review_type <<Array>>
user_review_type <<Multiset>>
external_review_type <<udt>> review_source review_ur
user_review_type <<udt>> review_date review_text reviewer score
products <<udt>> products
products <<persistent>>
product_type <<udt>> type_of_product_typ codeState id availability list_price production_compan release_date runtime format aspect_ratio dvd_encoding number_of_disc our_price movie getRating
product_type <<persistent>> <<Check>> type_of_product_typ <<NOT NULL>> type_of_product_type <<NOT NULL>> availability, <<PK>> codeState, id, <<NOT NULL>> movie, <<trigger>>getOur_price_inse <<trigger>>getOur_price_upda
Ref_product_type <<Ref>>
Ref_product_type <<Multiset>>
external_review_type <<persistent>>
user_review_type <<persistent>>

Figure D-3. OR Model for the OMDB Case Study (Diagrammer)

As well, Figure D-4 shows the same model displayed in the improved EMF tree-like editor.



```
platform:/resource/UML2SQL2003/Models/omdb/omdb.sql2003
  Schema Online Movie Database
    Structured Type people <<UDT>>
      Attribute persons : MULTISET (REF person_type) <<NOT NULL>> <<MULTISET>>
    Structured Type movies <<UDT>>
      Attribute movies : MULTISET (REF movie_type) <<NOT NULL>> <<MULTISET>>
    Structured Type movie_type <<UDT>>
      Attribute id : INTEGER <<PK>>
      Attribute studio : CHARACTERVARYING
      Attribute synopsis : CHARACTERVARYING
      Attribute title : CHARACTERVARYING <<NOT NULL>> <<UNIQUE>>
      Attribute website : CHARACTERVARYING
      Attribute genre : ARRAY [4] (CHARACTERVARYING) <<NOT NULL>> <<ARRAY>>
      Attribute ext_review : ARRAY [5] (external_review_type)  <<ARRAY>>
      Attribute user_review : MULTISET (user_review_type)  <<MULTISET>>
      Attribute products : MULTISET (REF product_type)  <<MULTISET>>
      Attribute directors : MULTISET (REF director_type) <<NOT NULL>> <<MULTISET>>
      Attribute playwriters : MULTISET (REF playwriter_type) <<NOT NULL>> <<MULTISET>>
      Attribute casts : MULTISET (REF cast_type) <<NOT NULL>> <<MULTISET>>
      Method numProductByYear (year : CHARACTERVARYING) : INTEGER
        Method Parameter year : CHARACTERVARYING
    Structured Type cast_type <<UDT>>
      Attribute id : INTEGER <<PK>>
      Attribute casting_order : INTEGER
      Attribute role : CHARACTERVARYING <<NOT NULL>>
      Attribute actor : REF actor_type <<NOT NULL>> <<REF>>
      Attribute movie : REF movie_type <<NOT NULL>> <<REF>>
    Structured Type castings <<UDT>>
      Attribute casts : MULTISET (REF cast_type)  <<MULTISET>>
    Structured Type products <<UDT>>
      Attribute products : MULTISET (REF product_type)  <<MULTISET>>
    Structured Type external_review_type <<UDT>>
      Attribute review_source : CHARACTERVARYING
      Attribute review_url : CHARACTERVARYING
    Structured Type user_review_type <<UDT>>
      Attribute review_date : DATE
      Attribute review_text : CHARACTERVARYING
      Attribute reviewer : CHARACTERVARYING
      Attribute score : INTEGER
    Structured Type director_type <<UDT>>
    Structured Type actor_type <<UDT>>
      Attribute casts : MULTISET (REF cast_type) <<NOT NULL>> <<MULTISET>>
    Structured Type playwriter_type <<UDT>>
    Structured Type person_type <<UDT>>
      Attribute id : INTEGER <<PK>>
      Attribute country : CHARACTERVARYING
      Attribute dob : DATE
      Attribute name : CHARACTERVARYING <<NOT NULL>>
      Attribute sex : CHARACTER
    Structured Type MERGE_playwriter_type_director_type_actor_type <<UDT>>
    Structured Type product_type <<UDT>>
      Attribute type_of_product_type : CHARACTERVARYING <<NOT NULL>>
      Attribute codeState : INTEGER <<PK>>
      Attribute id : INTEGER <<PK>>
      Attribute availability : CHARACTERVARYING <<NOT NULL>>
      Attribute list_price : REAL
      Attribute production_company : CHARACTERVARYING
      Attribute release_date : DATE
      Attribute runtime : INTEGER {default=0}
      Attribute format : CHARACTERVARYING
      Attribute acpect_ratio : CHARACTERVARYING
      Attribute dvd_encoding : CHARACTERVARYING
      Attribute number_of_discs : INTEGER
      Attribute our_price : REAL
      Attribute movie : REF movie_type <<NOT NULL>> <<REF>>
      Method getRating () : CHARACTERVARYING
```

```
    ARRAY genre [4] : CHARACTERVARYING <<ARRAY>>
    ARRAY external_review_type [5] : external_review_type <<ARRAY>>
    Reference Type Ref_cast_type (cast_type) <<REF>>
    MULTISET Ref_cast_type : REF cast_type <<MULTISET>>
    Reference Type Ref_product_type (product_type) <<REF>>
    MULTISET Ref_product_type : REF product_type <<MULTISET>>
    MULTISET user_review_type : user_review_type <<MULTISET>>
    Reference Type Ref_actor_type (actor_type) <<REF>>
    Reference Type Ref_movie_type (movie_type) <<REF>>
    Reference Type Ref_director_type (director_type) <<REF>>
    MULTISET Ref_director_type : REF director_type <<MULTISET>>
    Reference Type Ref_playwriter_type (playwriter_type) <<REF>>
    MULTISET Ref_playwriter_type : REF playwriter_type <<MULTISET>>
    Reference Type Ref_person_type (person_type) <<REF>>
    MULTISET Ref_person_type : REF person_type <<MULTISET>>
    MULTISET Ref_movie_type : REF movie_type <<MULTISET>>
    Typed Table people <<PERSISTENT>>
      NN Not Null (persons)
    Typed Table movies <<PERSISTENT>>
      NN Not Null (movies)
    Typed Table movie_type <<PERSISTENT>>
      NN Not Null (genre)
      NN Not Null (title)
      Primary Key PK_id (id)
      Unique Constraint AK_title (title)
      NN Not Null (directors)
      NN Not Null (playwriters)
      NN Not Null (casts)
    Typed Table cast_type <<PERSISTENT>>
      NN Not Null (role)
      Primary Key PK_id (id)
      NN Not Null (actor)
      NN Not Null (movie)
    Typed Table castings <<PERSISTENT>>
    Typed Table products <<PERSISTENT>>
    Typed Table external_review_type <<PERSISTENT>>
    Typed Table user_review_type <<PERSISTENT>>
    Typed Table director_type <<PERSISTENT>>
      NN Not Null (name)
      Primary Key PK_id (id)
    Typed Table actor_type <<PERSISTENT>>
      NN Not Null (name)
      Primary Key PK_id (id)
      NN Not Null (casts)
    Typed Table playwriter_type <<PERSISTENT>>
      NN Not Null (name)
      Primary Key PK_id (id)
    Typed Table MERGE_playwriter_type_director_type_actor_type <<PERSISTENT>>
      NN Not Null (name)
      Primary Key PK_id (id)
    View GET ALL person_type <<VIEW>>
    Typed Table product_type <<PERSISTENT>>
      Trigger getOur_price_insert [AFTER INSERT]
      Trigger getOur_price_update [AFTER UPDATE]
      Table Check Constraint Check_Discriminant_type_of_product_type (type_of_product_type)
      NN Not Null (type_of_product_type)
      NN Not Null (availability)
      Primary Key PK_codeState_id (codeState, id)
      NN Not Null (movie)
```

Figure D-4. OR Model for the OMDB Case Study (improved EMF tree-like editor)

Bibliography

References

1. Acceleo. (2008). Acceleo: MDA generator - Home. Retrieved 16 October 2008, from http://www.acceleo.org/pages/home/en.
2. Achilleos, A., Georgalas, N., & Yang, K. (2007). *An Open Source Domain-Specific Tools Framework to Support Model Driven Development of OSS.* Paper presented at the European Conference on MDA (ECMDA-FA 2007), Haiffa, Israel.
3. Achilleos, A., Kun, Y., Georgalas, N., & Azmoodech, M. (2008). *Pervasive Service Creation using a Model Driven Petri Net Based Approach.* Paper presented at the International Conference on Wireless Communications and Mobile Computing Conference. IWCMC '08, Crete, Greece.
4. Acerbis, R., Bongio, A., Butti, S., Ceri, S., Ciapessoni, F., Conserva, C., et al. (2004). *WebRatio, an Innovative Technology for Web Application Development.* Paper presented at the Innternational Conference on Web Engineering (ICWE 2004), Munich, Germany.
5. Acerbis, R., Bongio, A., Brambilla, M. & Butti, S. (2007). *WebRatio 5: An Eclipse-Based CASE Tool for Engineering Web Applications*, in Web Engineering, 2007, 501-505.
6. Acerbis, R., Bongio, A., Brambilla, M., Butti, S., Ceri, S. & Fraternali, P. (2008). Web Applications Design and Development with WebML and WebRatio 5.0. In S. B. Heidelberg (Ed.), *Objects, Components, Models and Patterns. 46th International Conference, TOOLS EUROPE 2008, Zurich, Switzerland, June 30 - July 4, 2008. Proceedings* (Vol. 11, pp. 392-411).
7. ACR-NEMA (2003). *The DICOM Standard.* http://medical.nema.org/.
8. Acuña, C. Minoli, M. Vara, J.M. Model Driven Development of Semantic Web Services using Eclipse. 2009 Mexican International Conference on Computer Science, ENC'09. Mexico City, Mexico. (Submitted).
9. Agrawal, A. (2003). *Graph rewriting and transformation (GReAT): a solution for the model integrated computing (MIC) bottleneck.* Paper presented at the 18th IEEE International Conference on Automated Software Engineering (ASE 2003) Montreal, Canada.
10. Alder, G. (2003). *Design and Implementation of the JGraph Swing Component*. Retrieved August 15, 2003, from: www.jgraph.com.
11. Alder, G. (2003). *The JGraph tutorial*. Retrieved August 20, 2003 from: www.jgraph.com.
12. Amelunxen, C., Königs, A., Rötschke, T., & Schürr, A. (2006). *MOFLON: A Standard-Compliant Metamodelling Framework with Graph Transformations.* Paper presented at the Second European Conference on Model Driven Architecture – Foundations and Applications, ECMDA-FA 2006 Bilbao, Spain.
13. Allilaire, F. Jouault, F. *ATL Basic Examples and Patterns*. Retrieved 8 January, 2009, from: http://www.eclipse.org/m2m/atl/basicExamples_Patterns/.
14. Altova. (2008). XML Schema [Software]. Available from http://www.altova.com/dev_portal_xml_schema.html.
15. Ambler, S. (2006). Comparing the Various Approaches to Modelling in Software Development. Retrieved 23 August, 2007, from: http://www.agilemodeling.com/essays/modelingApproaches.htm
16. Amelunxen, C., Knigs, A., Rtschke, T., & Schrr, A. (2006). *MOFLON: A Standard-Compliant Metamodelling Framework with Graph Transformations.* Paper presented at the Model Driven Architecture - Foundations and Applications: Second European Conference (ECMDA-FA 2006), Bilbao, Spain.
17. AndroMDA. (2008). AndroMDA.org - Home. Retrieved 16 October 2008, from http://www.andromda.org/
18. Apache Software Foundation (2006). *The Apache Velocity Project*. Retrieved June, 23, 2007, from: http://velocity.apache.org/

19. Arsenault, S. (2007). Contributing Actions to the Eclipse Workbench. *Eclipse Corner Articles, January, 2007*. Retrieved from http://www.eclipse.org/articles/article.php?file=Article-action-contribution/index.html.
20. *ArcStyler 5.5*, Interactive Objects Software GmbH (iO GmbH). Freiburg, Germany.
21. Atkinson, C., & Kuhne, T. (2003). Model-driven development: a metamodelling foundation. *IEEE Software, 20*(5).
22. Atzeni, P. Ceri, S. Paraboschi, S. & Torlone, R. (1999). *Database Systems. Concepts, Languages and Architectures,* McGraw-Hill.
23. Atzeni, P., Cappellari, P., Torlone, R., Bernstein, P., A., & Gianforme, G. (2008). Model-independent schema translation. *The VLDB Journal, 17*(6), 1347-1370.
24. ATLANMOD. (2009). *Metamodel Zoos*. Retrieved, March 20, 2009, from http://www.emn.fr/x-info/atlanmod/index.php/Zoos.
25. Arora, R., Chu, D., Demirezen, Z., Gray, J., Gulotta, J., Pedro, L., Sanchez, A. Sullivan, G., & Ximing Y. Sematics Group Work Result at the 8th OOPSLA Workshop on Domain-Specific Modeling [PowerPoint Slides]. Retrieved December, 12, 2008 from http://www.dsmforum.org/events/DSM08/Slides/Group_on_Semantics_for_DSL.ppt
26. Avison, D., Lan, F., Myers, M., Nielsen, A. (1999). Action Research. *Communications of the ACM, 42*(1), 94-97.
27. Balasubramaniam, R., Curtis, S., Timothy, P., & Michael, E. (1997). Requirements traceability: Theory and practice. *Annals of Software Engineering, 3*, 397-415.
28. Balogh, A., & Varro, D. (2006). *Advanced model transformation language constructs in the VIATRA2 framework*. Paper presented at the ACM symposium on Applied computing (SAC 2006), Dijon, France.
29. Balser, M., Bäumler, S., Knapp, A., Reif, W., & Thums, A. (2004). Interactive Verification of UML State Machines. In *Formal Methods and Software Engineering* (pp. 434-448).
30. Barbero, M., Jouault, F., & Bezivin, J. (2008). *Model Driven Management of Complex Systems: Implementing the Macroscope's Vision.* Paper presented at the Engineering of Computer Based Systems, 2008. ECBS 2008. 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS), Belfast, UK.
31. Baresi, L., Garzotto, F., & Paolini, P. (2001). *Extending UML for Modeling Web Applications.* Paper presented at the 34th Annual Hawaii International Conference on System Sciences (HICSS-34), Island of Maui, Hawaii.
32. Baresi, L., Colazzo, S., & Mainetti, L. (2005). *First experiences on constraining consistency and adaptivity of W2000 models*. Paper presented at 2005 ACM symposium on Applied computing (SAC 2005), Santa Fe, Nuevo Mexico (USA).
33. Batini, C., Ceri, S., & Navathe, S. B. (1992). *Conceptual Database Design: An Entity-Relationship Approach*. Addison-Wesley.
34. Beeler, G. W., & Gardner, D. (2006). A Requirements Primer. *ACM Queue, 4*(7), 22-26.
35. Bernstein, P A. (2003). Applying Model Management to Classical Meta Data Problems. In *First Biennial Conference on Innovative Data Systems Research*, Asilomar, CA, USA.
36. Bernstein, P., A. , Halevy, A., Y., & Pottinger, R., A. (2000). A vision for management of complex models. *SIGMOD Records, 29*(4), 55-63.
37. Bertino, E. & Marcos, E. Object Oriented Database Systems. In O. Díaz y M. Piattini (Eds.), *Advanced Databases: Technology and Design*. Norwood, MA: Artech House, 2000.
38. Bex, G. J., Neven, F., & Bussche, J. V. d. (2004). *DTDs versus XML schema: a practical study.* Paper presented at the 7th International Workshop on the Web and Databases: colocated with ACM SIGMOD/PODS 2004 (WedDB04), Paris, France.
39. Bézivin, J. (2001). *From Object Composition to Model Transformation with the MDA.* Paper presented at the 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems (TOOLS39), Washington DC, USA.
40. Bézivin, J., & Jouault, F. (2006, April, 2006). *KM3: a DSL for metamodel specification*. Paper presented at the 8th IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS 2006), Bologna, Italy.

41. Bézivin, J. (2004). In search of a Basic Principle for Model Driven Engineering. *Novatica/Upgrade, V*(2), 21-24.
42. Bézivin, J. (2005). On the unification power of models. *Journal of Software and Systems Modeling, 4*(2), 171-188.
43. Bézivin, J., Joualt, F., Rosenthal, P., & Valduriez, P. (2005). *Modeling in the large and modeling in the small.* Paper presented at the Model Driven Architecture, Europen MDA Workshops: Foundations and Applications (MDA-FA: 2003-2004), Twente, The Netherlands.
44. Bézivin, J., Hillairet, G., Jouault, F., Piers, W., & Kurtev, I. (2005). *Bridging the MS/DSL Tools and the Eclipse Modeling Framework.* Paper presented at the OOPSLA2005 International Workshop on Software Factories, San Diego, USA.
45. Bézivin, J., Jouault, F., Brunette, C., Chevrel, R., & Kurtev, I. (2005). *Bridging the Generic Modeling Environment (GME) and the Eclipse Modeling Framework (EMF).* Paper presented at the OOPSLA2005 International Workshop on Best Practices for Model Driven Software Development San Diego, California, USA.
46. Bézivin, J., Rumpe, B., Schürr, A., & Tratt, L. (2005). *Mandatory Example Specification*. CFP of the Model Transformations in Practice Workshop at MoDELS 2005, Montego Bay, Jamaica.
47. Bézivin, J. (2005). *Some Lessons Learnt in the Building of a Model Engineering Platform*. Paper presented at the 4th Workshop in Software Model Engineering (WISME), Montego Bay, Jamaica.
48. Bezivin, J., Jouault, F., Touzet, D. (2005). *An introduction to the ATLAS Model Management Architecture* (LINA Research Report Nº05.01). Nantes: University of Nantes. Retrieved June 20, 2007, from http://www.sciences.univ-nantes.fr/lina/atl/www/papers/RR-LINA2005-01.pdf.
49. Bézivin, J., Büttner, F., Gogolla, M., Jouault, F., Kurtev, I., & Lindow, A. (2006). *Model Transformations? Transformation Models!* Paper presented at the 9th International Conference on Model Driven Engineering Languages and Systems, MoDELS 2006, Genève, Italy.
50. Bezivin, J., Pierantonio, A. & Vallecillo, A. (2006). *Special track on model transformation (MT 2006).* In Proceedings of the 2006 ACM symposium on Applied computing, Dijon (France), 2006, pp. 1186-1187.
51. Bézivin, J., Bouzitouna, S., Del Fabro, M., Gervais, M. P., Jouault, F., Kolovos, D., et al. (2006). *A Canonical Scheme for Model Composition.* Paper presented at the European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA'06), Bilbao, Spain.
52. Bézivin, J., Vallecillo, A., García-Molina, J., & Rossi, G. (2008). MDA at the Age of Seven: Past, Present and Future. *UPGRADE, IX*(2), 4-7.
53. Biermann, E., Ermel, C., Hurrelmann, J., & Ehrig, K. (2008). *Flexible visualization of automatic simulation based on structured graph transformation.* Paper presented at the IEEE Symposium on Visual Languages and Human-Centric Computing, 2008. VL/HCC 2008., Herrsching am Ammersee, Germany.
54. Bitton, D., DeWitt, D., J. , & Turbyfill, C. (1983). *Benchmarking Database Systems A Systematic Approach.* Paper presented at the Proceedings of the 9th International Conference on Very Large Data Bases (VLDB 1983), Florence, Italy.
55. Blanc, X., Gervais, M.-P., & Sriplakich, P. (2005). *Model Bus: Towards the Interoperability of Modelling Tools.* Paper presented at the European MDA Workshop: Foundations and Applications, MDAFA 2004, Linköping, Sweden.
56. Bloch, J. (2008). *Effective JAVA (2nd. Edition)*: Prentice Hall.
57. Bohlen, M. (2006). QVT und Multi-Metamodell-Trans*formationen in MDA*. OBJEKTspektrum. Vol. 2 (March/April 2006). Translated in: http://www.andromda.org/jira/secure/attachment/10744/bohlen_OS_02_06_k4.pdf.
58. Booch, G., Brown, A. W., Iyengar, S., Rumbaugh, J., & Selic, B. (2004). An MDA Manifesto. *Business Process Trends/MDA Journal*.

59. Borland. (2008). Software Architecture Design, Visual UML & Business Process Modeling from Borland. Retrieved 16 October 2008, from http://www.borland.com/us/products/together/index.html
60. Boronat, A., Carsí, J., & Ramos, I. (2006). *Algebraic Specification of a Model Transformation Engine*. Paper presented at the Fundamental Approaches to Software Engineering (FASE'06), Vienna, Austria.
61. Boronat, A. (2007). Ph. D Thesis. *A formal framework for model management.* Technical University of Valencia, Valencia.
62. Braun, P., & Marschall, F. (2003). *The Bi-directional Object-Oriented Transformation Language* (No. TUM-I0307). Munich, Germany: Technische UniversitätMünchen.
63. Brooks, F. P. (1995). The Mythical Man-Month: Essays on Software Engineering, 20th Anniversary Edition: Addison-Wesley Professional.
64. Bruel, J-M. (ed.) (2006). *Model Transformations in Practice Workshop*. In Proceedings of Satellite Events at the MoDELS 2005 Conference, LNCS, Vol. 3844 (Springer, 2006).
65. Brun, C. (2007). EMF Compare: One year later. *Eclipse Summit Europe, Eclipse Modeling Symposium*. Ludwisburg, Germany. [PDF Document]. Retrieved from: http://www.eclipsecon.org/summiteurope2007/presentations/ESE2007_EMFCompare.pdf, May, 20, 2009.
66. Budinsky, F., Merks, E., & Steinberg, D. (2008). *Eclipse Modeling Framework 2.0 (2nd Edition)*: Addison-Wesley Professional.
67. Bunge, M. (1979). *La Investigación Científica*. Barcelona: Ariel.
68. Burmester, S., Giese, H., Niere, J., Tichy, M., Wadsack, J. P., Wagner, R., et al. (2004). Tool integration at the meta-model level: the Fujaba approach. *International Journal on Software Tools for Technology Transfer (STTT), 6*(3), 203-218.
69. Burstall, R. M., & John, D. (1977). A Transformation System for Developing Recursive Programs. *Journal of the ACM, 24*(1), 44-67.
70. Buttner, F., & Gogolla, M. (2004). *Realizing UML Metamodel Transformations with AGG*, In Proceedings of the Workshop on Graph Transformation and Visual Modelling Techniques (GT-VMT 2004), Barcelona, Spain.
71. Cabot, J., & Teniente, E. (2006). *Constraint Support in MDA Tools: A Survey*. Paper presented at the Second European Conference on Model Driven Architecture – Foundations and Applications (ECMDA-FA 2006), Bilbao, Spain.
72. Cabot, J., Clarisó, R., Guerra, E., & de Lara, J. (2008). *An Invariant-Based Method for the Analysis of Declarative Model-to-Model Transformations.* Paper presented at the 11th International Conference on Model Driven Engineering Languages and Systems, MoDELS 2008, Toulouse, France.
73. Cáceres, P., Marcos, E., Vela, B. A MDA-Based Approach for Web Information System Development, Proceedings of Workshop in Software Model Engineering.
74. Cáceres, P., De Castro, V., Vara, J. M., & Marcos, E. (2006, April 23 - 27, 2006). *Model Transformations for Hypertext Modeling on Web Information Systems.* Paper presented at the ACM Symposium on Applied computing (SAC), Dijon, France.
75. Cachero C., Melia S., Genero M., Poels G., Calero C. (2007). *Towards Improving the Navigability of Web Applications: A Model-Driven Approach*, in European Journal of Information Systems, 2007, Vol. 16, 420-447.
76. CARE Technoligies. (2009). OlivaNOVA [Software]. Available from http://www.care-t.com/
77. Carlson, D. (2001). Modeling XML Vocabularies with UML, Part I, II & III. *XML.com*. Retrieved from http://www.xml.com/pub/a/2001/08/22/uml.html.
78. Carlson, D. (2001). Modeling XML Applications with UML: Practical e-Business Applications: Addison-Wesley Professional.
79. Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., & Matera, M. (2002). *Designing Data-Intensive Web Applications*: Morgan Kaufmann Publishers Inc.
80. Chamberlin, D. (1996). Using the New DB2: IBM's Object-Relational Database System: Ap Professional.

81. Chaudhri, A.B., Rashid, A. & Zicari, R. (2003). *XML Data Management. Native XML and XML-Enabled Database Systems*: Addison-Wesley Professional.
82. Chen, P. P. 1994. The entity-relationship model—toward a unified view of data. *ACM Transactions on Database Systems, 1*(1), 9-36.
83. Chomsky, N. (1956). Three models for the description of language. *Information Theory, IEEE Transactions, 2*(3), 113-124.
84. Cicchetti, A., Ruscio, D. D., Eramo, R., & Pierantonio, A. (2008). *Automating Co-evolution in Model-Driven Engineering.* Paper presented at the 12th International IEEE Enterprise Distributed Object Computing Conference - EDOC 2008, München, Germany.
85. Clark T., Evans A., Sammut P., Willans J. (2004) *An eXecutable metamodelling facility for domain specific language design*. In Proceedings of the 4th OOPSLA Workshop on Domain-Specific Modeling, DSM'04. Tolvanen, Finland.
86. Clark, T., Evans, A., Sammut, P., & Willans, J. (2008). *Applied Metamodelling - A Foundation for Language Driven Development (2nd Edition)*. Available from http://itcentre.tvu.ac.uk/~clark/book.html
87. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., et al. (1999). *The Maude System.* Paper presented at the 10th International Conference on Rewriting Techniques and Applications, Trento, Italy.
88. *Codagen Architect*, Codagen Technologies Corp., Montreal, Canada.
89. Codd, E. F. (1983). A relational model of data for large shared data banks. *Communications of the ACM, 26*(1), 64-69.
90. Cook, S., Jones, G., Kent, S., & Cameron Wills, A. (2007). *Domain-Specific Development with Visual Studio DSL Tools* Addison-Wesley Professional.
91. Cook, S., & Kent, S. (2008). The Domain-Specific IDE. *UPGRADE, IX*(2), 17-21.
92. Computer Associates. AllFusion ERwin Data Modeler. http://www.ca.com/us/products/product.aspx?id=260.
93. Corradini, A., Ehrig, H., Löwe, M., Montanari, U., & Padberg, J. (1996). *The Category of Typed Graph Grammars and its Adjunctions with Categories.* Paper presented at the 5th International Workshop on Graph Gramars and Their Application to Computer Science, Williamsburg, VA, USA.
94. Csertan, G., Huszerl, G., Majzik, I., Pap, Z., Pataricza, A. & Varro, D. (2002). *VIATRA — Visual Automated Transformations for Formal Verification and Validation of UML Models*. In Proceedings of 17th IEEE International Conference on Automated Software Engineering (ASE'02), IEEE Computer Society, Los Alamitos, CA, USA, 2002.
95. Curran, J. (2003). *Conformance Testing: An Industry Perspective*. Java Conformance Testing Sun Microsystems.
96. Czarnecki, K., & Eisenecker, U. (1999). *Components and Generative Programming.* Paper presented at the 7th European Software Engineering Conference / 7th ACM SIGSOFT Symposium on the Foundations of Software Engineering — ESEC/FSE '99, Toulouse, France.
97. Czarnecki, K., & Helsen, S. (2003). *Classification of Model Transformation Approaches.* Paper presented at the 2nd OOPSLA Workshop on Generative Techniques in the Context of MDA, Anaheim, USA.
98. Czarnecki, K., & Helsen, S. (2006). Feature-based survey of model transformation approaches. *IBM Systems Jorunal, 45*(3), 621-645.
99. Czarnecki, K., Natahan foster, J., Hu, Z., Lämmel, R., Schürr, A. and Terwiligger, J. F. *Bidirectional Transformations: a cross-discipline perspective. GRACE meeting notes, state of the art and outlook*. Proceedings of the International Conference on Model-Transformations (ICMT 2009). 29-30 June 2009, Zurich (Switzerland).
100. Dalci, E., Fong, E., & Goldfine, A. (2003). *Requirements for GSC-IS Reference Implementations*: National Institute of Standards and Technology, Information Technology Laboratory.
101. Damus, C. (2007). Implementing Model Integrity in EMF with MDT OCL. *Eclipse Corner Articles, February 9, 2007.* Retrieved from

http://www.eclipse.org/articles/article.php?file=Article-EMF-Codegen-with-OCL/index.html
102. Damus, C. W. (2008). Object Constraint Language. In OMG (Ed.), *Symposium on Eclipse Open Source Software and OMG Open Specifications*. Ottawa, Ontario, Canada: OMG.
103. Davis, J. (2003). *GME: the generic modeling environment.* Paper presented at the Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. OOPSLA '03, Anaheim, CA, USA.
104. De Castro, V., Marcos, E., & Cáceres, P. (2004). A User Service Oriented Method to Model Web Information Systems. In *WISE* (Vol. 3306, pp. 41-52): Springer.
105. De Castro, V., Marcos, E., & Lopez Sanz, M. (2006). A model driven method for service composition modelling: a case study. *International Jorunal on Web Engineering Technology*, *2*(4), 335-353.
106. De Castro, V., Vara, J. M., & Marcos, E. (2007). *Model Transformation for Service-Oriented Web Applications Development.* Paper presented at the 3rd International Workshop on Model-Driven on Web Engineering (MDWE 2007), Como, Italy.
107. De Castro, V., Vara, J. M., Herrmann, E., & Marcos, E. (2008). *A Model Driven Approach for the Alignment of Business and Information Systems Models*. Paper presented at the Proceedings of the 2008 Mexican International Conference on Computer Science, ENC'08. Mexicali, Baja California, *Mexico*
108. De Castro, V., Vara, J. M., Herrmann, E., & Marcos, E. (2008, September 2008). *From Real Computational Independent Models to Information System Models: An MDE approach.* Paper presented at the 4th International Workshop on Model-Driven Web Engineering (MDWE 2008), Toulouse, France.
109. De Lara, J., Vangheluwe, H. & Alfonseca, M. (2004). *Meta-Modelling and Graph Grammars for Multi-Paradigm Modelling in AToM3*, Journal on Software and Systems Modelling, Vol 3(*3*).
110. Den Han, J. (2008, June 11). MDA *MDD MDE MDSD MDSE: help!*. Retrieved from http://www.theenterprisearchitect.eu/
111. Den Han, J. (2009, January 9). *MDE - Model Driven Engineering - reference guide*. Retrieved from http://www.theenterprisearchitect.eu/
112. Diaz, P., Montero, S. and Aedo, I. (2005). *Modelling Hypermedia and Web Applications: the Ariadne Development Method*, in Information Systems, Vol.30(8), 2005, 649-673.
113. Didonet Del Fabro, M., Bézivin, J., & Valduriez, P. (2006). *Model-Driven Tool Interoperability: An Application in Bug Tracking.* Paper presented at the OTM Confederated International Conferences, CoopIS, DOA, GADA, and ODBASE 2006, Montpellier, France.
114. Didonet Del Fabro, M., Bézivin, J. & Valduriez P. (2006). *Weaving Models with the Eclipse AMW plugin*. Eclipse Modeling Symposium, Eclipse Summit Europe, Esslingen, Alemania.
115. Didonet Del Fabro, M. (2007). *Metadata management using model weaving and model transformation*. Ph.D. Thesis University of Nantes, Nantes, France.
116. Dijkstra, E.W. *A Discipline of Programming*. Englewood Cliffs, NJ: Prentice Hall, 1976.
117. Drake, M. (2003). Oracle XML DB White Paper. Oracle Corporation.
118. Dobing, B., & Parsons, J. (2006). How UML is used. *Communications of the ACM, 49*(5), 109-113.
119. Eckerson, Wayne W. Three Tier Client/Server Architecture: Achieving Scalability, Performance, and Efficiency in Client Server Applications. *Open Information Systems 10* (3), 1-20.
120. Eclipse Foundation. (2009). *GMF Tutorial*. Retrieved November, 2008, from: http://wiki.eclipse.org/GMF_Tutorial.
121. Eckel, B. (1998). *Thinking in JAVA* (1st. Edition): Prentice Hall.
122. Efftinge, S., Völter, M., Haase, A., & Kolb, B. (2006). The Pragmatic Code Generator Programmer. *TheServerSide.com, September, 2006*. Retrieved from http://www.theserverside.com/tt/articles/article.tss?l=PragmaticGen.

123. Efftinge, S. (2006). openArchitectureWare 4.1 Xtend language reference. Retrieved 22 July, 2007, from http://www.eclipse.org/gmt/oaw/doc/4.1/r25_extendReference.pdf
124. Efftinge, S. (2006). openArchitectureWare 4.1 Xtext language reference. Retrieved 22 July, 2007, from http://www.eclipse.org/gmt/oaw/doc/4.1/r80_xtextReference.pdf
125. Efftinge, S., Friese, P., & Köhnlein, J. (2008). Best Practices for Model-Driven Software Development. *InfoQ*. Retrieved from http://www.infoq.com/articles/model-driven-dev-best-practices.
126. Ehrig, H., Engels, G., Kreowski, H.-J., Rozenberg, G. 1999. *Handbook of Graph Grammars and Computing by Graph Transformation.Vol(1)*. World Scientific.
127. eXcelon Corporation. *Managing DXE. System Documentation Release 3.5.* eXcelon Corporation. Burlington. Retrieved from: www.excelon.corp.com, 2003.
128. EMC^2 Corporation. X-Hive/DB 8.0. http://www.x-hive.com/.
129. Engstrom, E., & Krueger, J. (2000). *Building and rapidly evolving domain-specific tools with DOME.* Paper presented at the IEEE International Symposium on Computer-Aided Control System Design. CACSD 2000, Anchorage, Alaska, USA
130. Estévez, A., Padrón, J., Sánchez Rebull, V., & Roda, J. L. (2006). *ATC: A Low-Level Model Transformation Language.* Paper presented at the II International Workshop on Model-Driven Enterprise Information Systems 2006 (MDEIS 2006) in 8th International Conference on Enterprise Information Systems (ICEIS) 2006, Pahos, Chipre.
131. Favre, J. (2004). *Towards a Basic Theory to Model Model Driven Engineering.* Paper presented at the Workshop on Software Model Engineering, WISME 2004, joint event with UML2004, Lisbon, Portugal.
132. Ferrante, L. (1987). A comparison of the ISO working draft standard for SQL and a commercial implementation of SQL. *SIGSMALL/PC Notes, 13*(3), 28-55.
133. Feuerlicht, G., Pokorný, J., & Richta, K. (2009). *Object-Relational Database Design: Can Your Application Benefit from SQL:2003?* Paper presented at the 16th International Conference on Information Systems Development (ISD2007), Galway, Ireland.
134. Flore, F. (2003). MDA: The Proof is in Automating Transformations between Models. OptimalJ White Paper, pp. 1–4.
135. Fondemont, F. (2007). *Concrete syntax definition for modeling languages.* Ph.D. Thesis. Ecole polytechnique fédérale de Lausanne EPFL, Lausanne.
136. Fons, J., Pelechano, V., Pastor, O., Valderas, P., Torres, V. (2007). *Applying the OOWS Model-Driven Approach for Developing Web Applications. The Internet Movie Database Case Study*, in Web Engineering: Modelling and Implementing Web Applications. Springer Human-Computer Interaction Series. Rossi, G.; Pastor, O.; Schwabe, D.; Olsina, L. (Eds.), 2007.
137. Fowler, M. (2004). UML Distilled: A Brief Guide to the Standard Object Modeling Language (3rd. Edition). Boston, MA: Addison-Wesley.
138. Fowler, M. (2005). Language Workbenches: The Killer-App for Domain Specific Languages?. Retrieved from http://martinfowler.com/articles/languageWorkbench.html.
139. Fowler, M. (2005). Language Workbenches and Model-Driven Architecture. Retrieved from http://martinfowler.com/articles/mdaLanguageWorkbench.html.
140. Fowler, M. (2009). Code Generation for Dummies. *Methods and Tools*, Spring 2009, 65-82.
141. France Telecom, F. (2008). SmartQVT - A QVT implementation. Retrieved 16 October 2008, from http://smartqvt.elibel.tm.fr/
142. France, R. B., Ghosh, S., Dinh-Trong, T., & Solberg, A. (2006). Model-driven development using UML 2.0: promises and pitfalls. *Computer, 39*(2), 59-66.
143. Frankel, D. (2002). Model Driven Architecture: Applying MDA to Enterprise Computing. . New York, USA: John Wiley & Sons.
144. French, W.L. & Bell, C.H. Jr. *Desarrollo organizacional* (5ª ed.). Naucalpán de Juárez, México, Prentice-Hall, 1996.
145. Frederick, S., Dan, G., Greg, M., Luke, H., & Trevor, J. (2003). Compiling for template-based run-time code generation. *Jorunal of Functional Programming, 13*(3), 677-708.

146. Friese, P., Efftinge, S., & Köhnlein, J. (2008). Build your own textual DSL with Tools from the Eclipse Modeling Project. *Eclipse Corner Articles*. Retrieved from http://www.eclipse.org/articles/article.php?file=Article-BuildYourOwnDSL/
147. Fuentes, L. & Vallecillo, A. (2004). An introduction to UML profiles. *UPGRADE, European Journal for the Informatics Professional*, 5(2):5-13.
148. Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1994). *Design Patterns, Elements of Reusable Object-Oriented Software*: Addison-Wesley.
149. García, M. (2007, November, 28). What QVT for Eclipse has done to me. Message posted to Eclipse M2M newsgroup (http://dev.eclipse.org/newslists/news.eclipse.modeling.m2m).
150. Garcia, M., & Sentosa, P. (2008). *Generation of Eclipse-based IDEs for Custom DSLs*. Hamburg, Germany: Software Systems Institute (STS), Technische Universit at Hamburg-Harburg.
151. Garzotto, F., Paolini, P., & Schwabe, D. (1993). HDM- a Model-Based Approach to Hypertext Application Design. *ACM Transaction On Database Systems, 11*(1), 1-26.
152. GEMS Project. http://www.eclipse.org/gmt/gems/
153. Gerber, A., Lawley, M., Raymond, K., Steel, J., & Wood, A. (2002). Transformation: The missing link of MDA. *Graph Transformation: First International Conference, ICGT 2002*, 90-105.
154. Gill, T., Gilliland-Swetland, A., & Baca, M. (2000). *Introduction to Metadata: Pathways to Digital Information*. Los Angeles, USA: Getty Information Institute.
155. Glineur, Q. M2M/Relational QVT Language (QVTR). Retrieved March, 2009, from: http://wiki.eclipse.org/M2M/Relational_QVT_Language_(QVTR).
156. Gómez, J., & Cachero, C. (2003). OO-H Method: extending UML to model web interfaces. In *Information modeling for internet applications* (pp. 144-173): IGI Publishing.
157. Guttman, M., & Parodi, J. (2006). Real-Life MDA: Solving Business Problems with Model Driven Architecture Morgan Kaufmann.
158. Goldschmidt, T., Becker, S., & Uhl, A. (2008). *Classification of Concrete Textual Syntax Mapping Approaches.* Paper presented at the 4th European Conference on Model Driven Architecture - Foundations and Applications, ECMDA-FA 2008, Berlin, Germany.
159. Gómez, J., Bia, A. & Parraga, A. (2005). *Tool Support for Model-Driven Development of Web Applications*, in Web Information Systems Engineering (WISE 2005), 2005, 721-730.
160. Gonzalez-Perez, C., & Henderson-Sellers, B. (2007). Modelling software development methodologies: A conceptual foundation. *Journal of Systems and Software, 80*(11), 1778-1796.
161. Gronback, R. C. (2009). Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit: Addison-Wesley Professional.
162. Grose, T. J., Doney, G. C., & Brodsky, S. A. (2002). *Mastering XMI: Java Programming with XMI, XML, and UML*: Wiley.
163. Grunske, L., Geiger, L., & Lawley, M. (2005). *A Graphical Specification of Model Transformations with Triple Graph Grammars.* Paper presented at the First European Conference on Model Driven Architecture – Foundations and Applications (ECMDA-FA 2005), Nuremberg, Germany.
164. Gurevich, Y. (2000). Sequential abstract-state machines capture sequential algorithms. *ACM Transactions on Computational Logic, 1*(1), 77-111.
165. Guttman, M., & Parodi, J. (2006). *Real-Life MDA: Solving Business Problems with Model Driven Architecture* Morgan Kaufmann.
166. Hailpern, B., & Tarr, P. (2006). Model-driven development: The good, the bad, and the ugly. *IBM Systems Journal, 45*(3), 451-461.
167. Holt, R. C., Winter, A. and Schürr, A. (2000) GXL: Toward a Standard Exchange Format. *Seventh Working Conference on Reverse Engineering*. IEEE Computer Society. Los Alamitos, USA.
168. IBM Corporation (2004). *DB2 9 pureXML*..
169. IBM Corporation. (2004). Informix Dynamic Server. http://www-01.ibm.com/software/data/informix/ids/

170. IBM. (2004). IBM Model Transformation Framework 1.0.2 Programmer's Guide. Retrieved, July 23, 2006, from: http://www.alphaworks.ibm.com/tech/mtf
171. IBM DB2 Universal Database. http://www-306.ibm.com/software/data/db2/
172. IBM. (2004). Emfatic. http://www.alphaworks.ibm.com/tech/emfatic. Retrieved, October 2006.
173. IBM. (2009). Rational Rose Product line. Retrieved 20 January, 2009, from: http://www-01.ibm.com/software/awdtools/developer/rose/index.html
174. ikv++ technologies. (2008). medini QVT. Retrieved 16 October 2008, from http://www.ikv.de/index.php?option=com_content&task=view&id=75&Itemid=77&lang=en.
175. IRISA (2009). Kermeta workbench: http://www.kermeta.org/.
176. ISO (International Standards Organization for Standardization) & IEC (International Electrotechnical Commission) (2003). *ISO/IEC 9075:2003 Information technology – Database languages – SQL:2003.*
177. Jacobson, I., Booch, G., & Rumbaugh, J. (1999). *The Unified Software Development Process* Addison-Wesley Professional.
178. JBoss, a division of Red Hat. Hibernate. http://www.hibernate.org/
179. Jézéquel, J-M. *Model Transformation Techniques*. Rennes: France: IRISA-Universite de Rennes. Retrieved December, 2007, from: http://modelware.inria.fr/rubrique21.html.
180. Jouault, F., Laarman, A., Allilaire, F. & Glineur, Q. (2005). Specification of the ATL Virtual Machine. Retrieved 20 January, 2009, from: http://www.eclipse.org/m2m/atl/doc/ATL_VMSpecification[v00.01].pdf
181. Jouault, F., & Bézivin, J. (2006). *KM3: A DSL for Metamodel Specification*. Paper presented at the 8th IFIP WG 6.1 International Conference on Formal Methods for Open Object-Based Distributed Systems, FMOODS 2006, Bologna, Italy.
182. Jouault, F., & Kurtev, I. (2006). *On the architectural alignment of ATL and QVT.* Paper presented at the ACM symposium on Applied computing (SAC 2006), Dijon, France.
183. Jouault, F., Bezivin, J. and Kurtev, I. (2006). *TCS: a DSL for the Specification of Textual Concrete Syntaxes in Model Engineering*. In: GPCE'06: Proceedings of the fifth international conference on Generative programming and Component Engineering, Portland, Oregon, USA, pages 249--254.
184. Jouault, F., & Kurtev, I. (2006). Transforming Models with ATL. In *Satellite Events at the MoDELS 2005 Conference* (pp. 128-138).
185. Jouault, F. & Piers, W. (2009). *ATL User Guide*. Retrieved 15 January, 2009, from http://wiki.eclipse.org/ATL/User_Guide.
186. Kalnins, A., Celms, E., & Sostaks, A. (2006). *Model Transformation Approach Based on MOLA.* Paper presented at the Model Transformations in Practice Workshop - Satellite Events at the MoDELS 2005 Conference, Montego Bay, Jamaica.
187. Kelly, S., Lyytinen, K., & Rossi, M. (1996). MetaEdit+ A fully configurable multi-user and multi-tool CASE and CAME environment. In *8th International Conference on Advanced Information Systems Engineering, CAiSE'96* (pp. 1-21). Heraklion, Crete: Springer.
188. Kelly, S. (2005, May 03). *XMI, MOF and MetaEdit+*. Retrieved from: http://www.metacase.com/blogs/stevek/
189. Kelly, S., & Tolvanen, J.-P. (2008). *Domain-Specific Modelling: Enabling Full Code Generation*: Wiley-IEEE Computer Society Press.
190. Kelly, S. (2008, September). *DSM in the solution domain?* [Msg 7]. Message posted to http://www.linkedin.com/groups?home=&gid=138803&trk=anet_ug_hm
191. Kern, H. (2008). *The Interchange of (Meta)Models between MetaEdit+ and Eclipse EMF.* Paper presented at the 8th ooPSLA Workshop on Domain-Specific Modelling at OOPSLA 2008, Birmingham, USA.
192. Klatt, B. (2007). Xpand: A Closer Look at the model2text Transformation Language. Retrieved 10/16/2008, from http://www.bar54.de/publikationen.html.
193. Kleppe, A., Warmer, J., & Bast, W. (2003). MDA Explained: The Model Driven Architecture: Practice and Promise: Addison-Wesley.

194. Klimavicius, M., & Sukovskis, U. (2008). *Applying MDA and universal data models for data warehouse modeling.* Paper presented at the Proceedings of the 10th WSEAS International Conference on Automatic Control, Modelling & Simulation, Istanbul, Turkey.
195. Knapp, A., Koch, N., Moser, F., & Zhang, G. (2003). *ArgoUWE: A Case Tool for Web Applications.* Paper presented at the First International Workshop on Engineering Methods to Support Information Systems Evolution (EMSISE'03), Geneva, Switzerland.
196. Knap, A., Koch, N., Zhang, G. and Hassler, H.-M. (2004). *Modeling Business Processes in Web Applications with ArgoUWE*, in – Proc of Int. Conf. Unified Modeling Language (UML 2004), Springer LNCS 3273, 69-83.
197. Koch, N, Meliá, S. Moreno, N. Pelechano, V. Sanchez, F. & Vara, J.M. (2008). Model-Driven Web Engineering, *Novática-Upgrade Journal (English and Spanish), IX* (2), ISSN 1684-5285, Council of European Professional Informatics Societies (CEPIS), April 2008.
198. Koch, N. (2001). *Software Engineering for Adaptative Hypermedia Applications*. PhD Thesis, FAST Reihe Softwaretechnik Vol(12), Uni-Druck Publishing Company, Munich. Germany.
199. Koch, N. (2006). *Transformation Techniques in the Model-Driven Development Process of UWE*. In Workshop Proc. of the 6th Int. Conf. on Web Engineering (ICWE 2006), ACM Vol. 155, Palo Alto, California, 2006.
200. Koch, N., Meliá, S., Moreno, N., Pelechano, V., Sánchez, F., & Vara, J. M. (2008). Model-Driven Web Engineering. *UPGRADE, IX*(2), 40-45.
201. Kolovos, D., Paige, R., & Polack, F. (2006). Eclipse Development Tools for Epsilon, *Eclipse Summit Europe, Eclipse Modeling Symposium*. Esslingen, Germany.
202. Kolovos, D., Paige, R., & Polack, F. (2006). *The Epsilon Object Language (EOL).* Paper presented at the Model Driven Architecture - Foundations and Applications: Second European Conference (ECMDA-FA 2006), Bilbao, Spain.
203. Kolovos, D., Paige, R., Rose, L., & Polack, F. (2008). The Epsilon Book. Retrieved 20 December, 2008, from: http://epsilonlabs.svn.sourceforge.net/svnroot/epsilonlabs/org.eclipse.epsilon.book/Epsilon Book.pdf
204. Königs, A., & Schürr, A. (2006). Tool Integration with Triple Graph Grammars - A Survey. *Electronic Notes in Theoretical Computer Science, 148*(1), 113-150.
205. Kraus, A. (2008). *Model Driven Software Engineering for Web Applications.* Ph. D. Thesis. Ludwig-Maximilians-University München, Munich.
206. Kroib, C. & Koch, N. (2008). *UWE Metamodel and Profile: User Guide and Reference* (No. 0802). Munich, Germany: Programming and Software Engineering Unit (PST), Institute for Informatics, LMU – Ludwig-Maximilians-Universität. Retrieved November 18, 2008, from http://www.pst.ifi.lmu.de/projekte/uwe/
207. Kulkarni, V. & Reddy, S. (2003). Separation of Concerns in Model-Driven Development. *IEEE Software, 20*(5), 64-69.
208. Kurtev, I., Bezivin, J., & Aksit, M. (2002). *Technological Spaces: An Initial Appraisal.* Paper presented at the Confederated International Conferences DOA, CoopIS and ODBASE 2002, Industrial Track, Irvine, California, USA.
209. Kurtev, I. (2005). *Adaptability of model transformations.* Ph.D. Thesis. University of Twente, Enschede. Retrieved from http://purl.org/utwente/50761.
210. Kurtev, I. (2008). *State of the Art of QVT: A Model Transformation Language Standard.* Paper presented at the Third International Symposium on Applications of Graph Transformations with Industrial Relevance, AGTIVE 2007 Kassel, Germany.
211. Lawley, M., & Steel, J. (2006). Practical Declarative Model Transformation with Tefkat. In *Satellite Events at the MoDELS 2005 Conference* (pp. 139-150).
212. Lawley, M., & Raymon, K. (2007). *Implementing a practical declarative logic-based model transformation engine.* Paper presented at the ACM symposium on Applied computing 2007 (SAC 2007), Seoul, Korea.
213. Landin, P. J. (1965). Correspondence between ALGOL 60 and Church's Lambda-notation: Part I. *Communications of the ACM, 8*(2), 89-101.

214. Landin, P. J. (1965). Correspondence between ALGOL 60 and Church's Lambda-notations: Part II. *Communications of the ACM, 8*(3), 158-167.
215. Langlois, B., Jitia, C. E., and Jouenne, E. (2007). DSL Classification. In *Proceedings of the 7th OOPSLA Workshop on Domain-Specific Modeling*.
216. Ledeczi, A., Maroti, M., Bakay, A., Karsai, G., Garrett, J., Thomason, C., et al. (2001). *The Generic Modeling Environment.* Paper presented at the IEEE International Workshop on Intelligent Signal Processing (WISP'2001), Budapest, Hungary.
217. Lima, F., & Schwabe, D. (2003). *Modeling Applications for the Semantic Web* Paper presented at the International Conference on Web Engineering (ICWE 2003), Oviedo, Spain.
218. Lin, Y., Zhang, J., & Gray, J. (2005). A Testing Framework for Model Transformations. In *Model-Driven Software Development* (pp. 219-236): Springer.
219. Lin, Y., Strasunskas, D., Hakkarainen, S., Krogstie, J., & Solvberg, A. (2006). *Semantic Annotation Framework to Manage Semantic Heterogeneity of Process Models*. Paper presented at the 18th International Conference on Advanced Information Systems Engineering (CAISE 2006), Luxembourg, Luxembourg.
220. LMU – Ludwig-Maximilians-Universität München Institute for Informatics Programming and Software Engineering. MagicUWE – Reference. Retrieved 20 December, 2008, from http://www.pst.ifi.lmu.de/projekte/uwe/toolMagicUWEReference.html.
221. Lopez Sanz, M., Acuña, C. J., Cuesta, C. E., & Marcos, E. (2008). Modelling of Service-Oriented Architectures with UML. *Electronic Notes in Theoretical Computer Science, 194*(4), 23-37.
222. Lucas, F. J., & Toval, J. A. (2008). *Model Transformations powered by Rewriting Logic.* Paper presented at the CAiSE Forum at CAiSE'08, Montpellier, France.
223. Marcos, E. & Marcos, A. (1998). An Aristotelian Approach to the Methodological Research: a Method for Data Models Construction. In: *Information Systems - The Next Generation*. L. Brooks and C. Kimble (Eds.). McGraw-Hill. 1998.
224. Marcos, E., Vela, B., & Cavero, J. (2004). A methodological approach for object-relational database design using UML. *Informatik - Forschung und Entwicklung, 18*(3), 152-164.
225. Marcos, E., De Castro, V., Vela, B. (2004). Representing Web Services with UML: A Case Study, presented at First International Conference on Service Oriented Computing (ICSOC'03), Trento, Italy.
226. Marcos, E., Vela, B., & Cavero, J. M. (2001). Extending UML for Object-Relational Database Design. In *UML 2001 - The Unified Modeling Language. Modeling Languages, Concepts, and Tools*. 4th International Conference, Toronto, Canada, October 2001, Proceedings (Vol. 2185, pp. 225-239): Springer.
227. Marjan, M., Jan, H. & Anthony, M. S. (2005). When and how to develop domain-specific languages. *ACM Computer Surveys, 37*(4), 316-344.
228. Marković, S., & Baar, T. (2008). Refactoring OCL annotated UML class diagrams. *Software and Systems Modeling, 7*(1), 25-47.
229. Marschall, F., & Braun, P. (2003). *Model Transformations for the MDA with BOTL.* Paper presented at the Workshop on Model Driven Architecture: Foundations and Applications. MDAFA 2003., Enschede, The Netherlands.
230. Mayo Clinic. (2004) *Analyze Software*. http://www.mayo.edu/ bir/Software/ Analyze/ Analyze.html, 2004.
231. Mazón, J.-N., & Trujillo, J. (2008). An MDA approach for the development of data warehouses. *Decission Support Systems, 45*(1), 41-58.
232. McTaggart, R. (1991). Principles of Participatory Action Research. *Adult Education Quarterly, 41*(3).
233. Megginson Technologies Ltd. (2004). *SAX*. Retrieved July 30,2004, from: http://www.saxproject.org/.
234. Meliá, S., Gómez J. & Serrano J.L. (2007). *WebTE: MDA Transformation Engine for Web Applications*, in Proc. 7th Int. Conf. Web Engineering (ICWE 2007), Springer LCNS 4607, Como, Italy.

235. Meliá, S. & Gómez, J. (2006). The WebSA Approach: Applying Model-Driven Engineering to Web Applications. *Journal of Web Engineering*, *5*(2), 121–149.
236. Meliá, S., Gomez, J. (2006) *UPT. A Graphical Transformation Language based on a UML Profile*. In: Proceedings of European Workshop on Milestones, Models and Mappings for Model-Driven Architecture (3M4MDA 2006). 2nd European Conference on Model Driven Architecture (EC-MDA 2006), Bilbao, Spain.
237. Meliá, S., & Gómez, J. (2005). Applying Transformations to Model Driven Development of Web Applications. In S. B. Heidelberg (Ed.), *ER 2005 Workshops: Perspectives in Conceptual Modeling* (Vol. 3770/2005, pp. 63-73).
238. Mellor, S., & Balcer, M. (2002). *Executable UML: A Foundation for Model-Driven Architecture*. Indianapolis, IN: Addison-Wesley Professional.
239. Mellor, S., Scott, K., Uhl, A., & Weise, D. (2004). *MDA Distilled*. Paper presented at the Advances in Object-Oriented Information Systems: OOIS 2002 Workshops, Montpellier, France.
240. Mellor, S. (2006). Demystifying UML. *Embedded Systems Design*, *19*(3), 22-35.
241. Mernik, M., Heering, J., & Sloane, A. M. (2005). When and how to develop domain-specific languages. *ACM Computer Surveys*, *37*(4), 316-344.
242. Meservy, T. O., & Fenstermacher, K. D. (2005). Transforming software development: an MDA road map. *Computer*, *38*(9), 52-58.
243. MetaCase. MetaEdit+ [Software]. Available at http://www.metacase.com/mep/.
244. Microsoft Corporation (2000). Active Server Pages. Retrieved from http://msdn.microsoft.com/en-us/library/aa286483.aspx, October, 2007.
245. Microsoft Corporation. (2008). *Microsoft SQL Server 2008*. http://www.microsoft.com/spain/sql/
246. Miller, J., Mukerji, J. (Eds.), *MDA Guide Version 1.0*, OMG Document - omg/2003-05-01.
247. MOFLON development team. MOFLON website, 2007. available ay http://www.moflon.org.
248. Ministerio de Administraciones Públicas. *Metrica III*. Retrieved from: http://www.csi.map.es/csi/metrica3/, February, 2008.
249. Molina, F., Lucas, F. J., Toval, J. A., Vara, J. M., Cáceres, P., & Marcos, E. (2008). Toward Quality Web Information Systems Through Precise Model Driven Development. In C. Calero, M. A. Moraga & M. Piattini (Eds.), *Handbook of Research on Web Information Systems Quality* (pp. 344-363). Hershey PA, USA: Information Science Reference - IGI Global.
250. Moore, B., Dean, D., Gerber, A., Wagenknecht, G., & Vanderheyden, P. (2004). Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework: IBM.COM.
251. Moreno, N., Fraternali, P., & Vallecillo, A. (2007). WebML modelling in UML. *IET Software, 1*(3), 67-80.
252. Moreno, N., Romero, J., & Vallecillo, A. (2008). An overview of Model-Driven Web Engineering and the MDA. In *Web Engineering: Modelling and Implementing Web Applications* (pp. 353-382), London: Springer.
253. Moreno, N., & Vallecillo, A. (2008). Towards interoperable Web engineering methods. *Journal of American Society for Information Science and Technology*, *59*(7), 1073-1092..
254. Muller, P.-A., Fleurey, F., & Jézéquel, J.-M. (2005). *Weaving Executability into Object-Oriented Meta-languages.* Paper presented at the 8th International Conference on Model Driven Engineering Languages and Systems, MoDELS 2005, Montego Bay, Jamaica.
255. Muller, P.-A., Fleurey, F., Fondement, F., Hassenforder, M., Schneckenburger, R., Gérard, S., et al. (2006). *Model-Driven Analysis and Synthesis of Concrete Syntax.* Paper presented at the Model Driven Engineering Languages and Systems - MoDELS/UML 2006., Genova, Italy.
256. Murzek, M., & Kramler, G. (2007). *Business Process Model Transformation Issues - The Top 7 Adversaries Encountered at Defining Model Transformations.* Paper presented at the International Conference on Enterprise Information Systems (ICEIS 2007), Milan, Italy.

257. Musset, J. (2009). *A standard alternative for code generation: Acceleo MTL*. EclipseCON 2009, Santa Clara (CA), USA.
258. NetBeans. Metadata Repository (MDR) Project Home. Retrieved July 16, 2007, from: http://mdr.netbeans.org/
259. NoMagic Inc. (2009). MagicDraw UML [Software]. Available from http://www.magicdraw.com/.
260. Nunes, D. A., & Schwabe, D. (2006). *Rapid prototyping of web applications combining domain specific languages and model driven design.* Paper presented at the 6th International Conference on Web engineering (ICWE 2006), Palo Alto, California, USA.
261. Object Management Group (OMG). (2007). *Business Process Model and Notation (BPMN) 2.0 RFP*. OMG Document - BMI/2007-06-05.
262. Oldevik, J., Neple, T., Grønmo, R., Aagedal, J., & Berre, A.-J. (2005). Toward Standardised Model to Text Transformations. In *Model Driven Architecture – Foundations and Applications* (pp. 239-253).
263. OMG, *The CWM Specification*. OMG Document - ad/01-02-01, ad/01-02-02, ad/01-02-03.
264. OMG, Model Driven Architecture. A technical perspective. OMG document -ormsc/01-07-01
265. OMG. The Meta Object Facility (MOF) Core Specification, Version 2.0. OMG Document - formal/06-01-01
266. OMG, MOF Model to Text Transformation Language (MOFM2T), 1.0. OMG Document - formal/08-01-16
267. OMG, MOF Model to Text Transformation Language. RFP. Retrieved from http://www.omg.org/cgi-bin/doc?ad/04-04-07
268. OMG, Object Constraint Language Specification (OCL), version 2.0. OMG Document - formal/2006-05-01
269. OMG. Software Process Engineering Meta-Model (SPEM), version 2.0. OMG Document - formal/2008-04-01
270. OMG. *UML 2.1.1 Formal Specification*. OMG Document - formal/07-02-03.
271. OMG. *UML Diagram Interchange, v1.0.* OMG Document - formal/2006-04-04
272. OMG. *UML 1.5 Formal Specification*. OMG Document - formal/03-03-01.
273. OMG. MOF 2.0 Query/View/Transformation (QVT), V1.0. OMG Document - formal/08-04-03.
274. OMG. *MOF 2.0 Query/Views/Transformations RFP*, OMG document ad/2002-04-10 (2002).
275. OMG - XML Metadata Interchange (XMI) specification V2.1.1. OMG Document - formal/2007-12-01.
276. OMG. (2009). OMG Announces Model Interchange Working Group. Retrieved July, 20, from http://www.omg.org/news/releases/pr2009/07-08-09.htm.
277. openArchitectureWare (2008). *openArchitectureWare User guide (Version 4.3)* Retrieved February, 4, 2009, from http://www.openarchitectureware.org/pub/documentation/4.3.1/openArchitectureWare-4.3.1-Reference.pdf.
278. openMDX. (2008). openMDX - the leading open source MDA platform [Software]. Available at http://www.openmdx.org/
279. Oracle Corporation. *Oracle XML DB. Technical White Paper*. Retrieved from: www.otn.com, January, 2003.
280. Oracle Corporation. *Oracle10g. SQL Reference*, 2000.
281. Oracle Corporation. (2008). Oracle Designer 10g Release 2. [Software] Available from http://www.oracle.com/technology/products/designer/index.html
282. Oracle Corporation (2008). *Oracle Database 11g* [Software]. Available from http://www.oracle.com/global/lad/database/index.html.
283. Oxygen (2008). Oxygen XML Editor [Software]. Available from http://www.oxygenxml.com/xml_schema_editor.html.

284. Paiano, R., & Pandurino, A. (2004). *WAPS: Web Application Prototyping System.* Paper presented at the International Conference on Web Engineering, ICWE 2004, Munich, Germany.
285. Papazoglou, M., & van den Heuvel, W.-J. (2007). Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal The International Journal on Very Large Data Bases, 16*(3), 389-415.
286. Parnas, D. L. (1972). On the Criteria To Be Used in Decomposing Systems into Modules. *Communications of the ACM, 15*(12), 1053-1058.
287. Partsch, H., & Steinbrüggen, R. (1983). Program Transformation Systems. *ACM Computing Surveys, 15*(3), 199-236.
288. Pastor, O., Hayes, F., & Bear, S. (1992). *Oasis: An object-oriented specification language.* Paper presented at the 4th International Conference on Advanced Information Systems Engineering - CAiSE'92, Manchester, UK.
289. Pastor, O., Insfrán, E., Pelechano, V., Romero, J., & Merseguer, J. (1997). *OO-Method: An OO software production environment combining conventional and formal methods.* Paper presented at the Conference on Advanced Information Systems Engineering, CAiSE'97, Barcelona, Spain.
290. Patrascoiu, O. (2004). *YATL:Yet Another Transformation Language - Reference Manual Version 1.0*. Kent, Great Britain: Computing Laboratory, University of Kent. Retrieved from http://www.cs.kent.ac.uk/pubs/2004/1862
291. Philip, M., Schroeder, A., & Koch, N. (2008). *MDD4SOA: Model-Driven Service Orchestration.* Paper presented at the 12th International IEEE Enterprise Distributed Object Computing Conference - EDOC 2008, Munich, Germany.
292. PostgreSQL Global Development Group. PostgreSQL Database. http://www.postgresql.org/
293. Powell, A. (2004). *Generate code with Eclipse's Java Emitter Templates.* IBM developerWorks.
294. Quest Software. TOAD (TOol for Application Developers) [Software]. Available from http://www.toadsoft.com/
295. Ravi, M. & Sandeepan, B. (2003). *XML schemas in Oracle XML DB.* Paper presented at the Proceedings of the 29th international conference on Very Large Data Bases (VLDB 2003), BErling, Germany.
296. Reddy, S., Mulani, J., & Bahulkar, A. (2000). *Adex - a meta modeling framework for repository-centric systems building.* Paper presented at the International Conference on Advances in Data Management (COMAD-2000), Pune, India.
297. Robbins, J. E., & Redmiles, D. F. (2000). Cognitive support, UML adherence, and XMI interchange in Argo/UML. *Information and Software Technology, 42*(2), 79-89.
298. Roldán, V., Sánchez-Barbudo, A., & Ávila-García, O. (2008). Overcoming the Difficulties of Implementing a QVT Execution Solution. In OMG (Ed.), *Symposium on Eclipse Open Source Software and OMG Open Specifications*. Ottawa, Ontario, Canada: OMG Document: omg/08-06-47.
299. Romero, J. R., Rivera, J. E., Durán, F. & Vallecillo, A. (2007). Formal and Tool Support for Model Driven Engineering with Maude. *Journal of Object Technology, 6*(9), 187-207.
300. Royce, W. W. (1987). *Managing the development of large software systems: concepts and techniques.* Paper presented at the 9th international conference on Software Engineering, Monterey, California, United States.
301. Ruby, S., Thomas, D., & Heinemier Hansson, D. (2009). *Agile Web Development with Rails* (Third ed.): The Pragmatic Bookshelf.
302. Ruiz, M., Valderas, P. & Pelechano, V. (2007). Providing Methodological Support to Incorporate Presentation Properties in the Development of Web Services. In *E-Commerce and Web Technologies: 8th International Conference, EC-Web 2007* (Vol. 4655/2007, pp. 139-148). Regensburg, Germany: Springer Berlin / Heidelberg.
303. Saavedra, R., H. , & Smith, A., J. (1996). Analysis of benchmark characteristics and benchmark performance prediction. *ACM Transactions on Computer Systems, 14*(4), 344-384.

304. Salay, R., Chechik, M., Easterbrook, S., Diskin, Z., McCormick, P., Nejati, S., et al. (2007). *An Eclipse-based tool framework for software model management.* Paper presented at the OOPSLA 2007 - Workshop on Eclipse Technology eXchange - Eclipse'07, Montreal, Quebec, Canada.
305. Sánchez, V. (2008). Making a case for supporting a byte-code model transformation approach. In OMG (Ed.), *Symposium on Eclipse Open Source Software and OMG Open Specifications*. Ottawa, Ontario, Canada: OMG Document omg/08-06-48.
306. Sanchez-Barbudo, A., Sanchez, V., Roldán, V., Estévez, A., & Roda-García, J. L. (2008). *Providing an Open Virtual-Machine-based QVT Implementation.* Paper presented at the V Taller sobre Desarrollo de Software Dirigido por Modelos (DSDM 2008) in XII Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2008), Gijon, Spain.
307. Sánchez Cuadrado, J., García Molina, J., & Menarguez Tortosa, M. (2006). *RubyTL: A Practical, Extensible Transformation Language.* Paper presented at the European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA 2006), Bilbao, Spain.
308. Sánchez Cuadrado, J., & García Molina, J. (2007). Building Domain-Specific Languages for Model-Driven Development. *IEEE Software, 24*(5), 48-55.
309. Sánchez Cuadrado, J., & García Molina, J. (2008). *Approaches for Model Transformation Reuse: Factorization and Composition.* Paper presented at the 1st International conference on Theory and Practice of Model Transformations (ICMT 2008), Zurich, Switzerland.
310. Schauerhuber, A., Wimmer, M., & Kapsammer, E. (2006). *Bridging existing Web modeling languages to model-driven engineering: a metamodel for WebML.* Paper presented at the Second international workshop on model driven web engineering (MDWE'06) at the sixth International Conference on Web engineering (ICWE), Palo Alto, California.
311. Scheidgen, M. (2007, 27 June). Problems for Textual Model Notations. Message posted to http://metabubble.blogspot.com/.
312. Scheidgen, M. (2008). *Textual Modelling Embedded into Graphical Modelling.* Paper presented at the 4th European Conference on Model Driven Architecture – Foundations and Applications, ECMDA-FA 2008, Berlin, Germany.
313. Schürr, A. (1995). *Specification of Graph Translators with Triple Graph Grammars.* Paper presented at the 20th International Workshop on Graph-Theoretic Concepts in Computer Science (WG`94), Herrsching, Germany.
314. Schwabe, D., & Rossi, G. (1998). An Object Oriented Approach to Web-Based Application Design. *Theory and Practice of Object Systems, 4*(4), 207-225.
315. Seidewitz, E. (2003). What models mean. *IEEE Software, 20*(5).
316. Selic, B. (2003). The pragmatics of Model-Driven development, *IEEE Software, 20*(5), 19-25.
317. Selic, B. (2005) *What's new in UML 2.0?*. IBM Rational Software. April, 2005.
318. Selic, B. (2008). MDA Manifestations. *UPGRADE, IX*(2), 12-16.
319. Selic, B. (2008). Personal reflections on automation, programming culture, and model-based software engineering. *Automated Software Engineering, 15*(3), 379-391.
320. Sendall, S., & Kozaczynski, W. (2003). Model Transformation–the Heart and Soul of Model-Driven Software Development. *IEEE Software, 20*(5), 42-45.
321. Skinner, C. (2008, June 25). DSL + UML = Pragmatic Modeling. Retrieved from http://blogs.msdn.com/camerons/default.aspx.
322. Slack, S. E. (2008). The business analyst in model-driven architecture. *IBM developerWorks*. Retrieved from http://www.ibm.com/developerworks/architecture/library/ar-bamda/
323. Smolander, K., Lyytinen, K., Tahvanainen, V.-P., & Marttiin, P. (1991). MetaEdit— A flexible graphical environment for methodology modelling. In *Third International Conference on Advanced Information Systems Engineering. CAISE'91* (pp. 168-193). Trondheim, Norway: Springer.
324. Sodius. (2008). MDworkbench [Software] Available from http://www.mdworkbench.com/.

325. Software AG. (2008). *Tamino XML Server* [Software]. Available from http://www.softwareag.com/tamino/.
326. Solmi, R. (2008). The Whole Platform: a Language Workbench for Eclipse. On *EclipseCON 2008*. Santa Clara, California (USA).
327. Sparx Systems. Enterprise Architect 7.1. http://www.sparxsystems.com.au/products/ea/index.html
328. Stahl, T., Volter, M., & Czarnecki, K. (2006). Model-Driven Software Development: Technology, Engineering, Management: John Wiley & Sons.
329. Steel, J., & Jézéquel, J.-M. (2007). On Model Typing. *Journal of Software and Systems Modeling (SoSyM), 6*(4), 401-414.
330. Stevens, P. (2008). *A Landscape of Bidirectional Model Transformations.* Paper presented at the International Summer School on Generative and Transformational Techniques in Software Engineering II, GTTSE 2007, Braga, Portugal.
331. Stonebraker, M. & Brown, P. (1999). *Object-Relational DBMSs. Tracking the Next Great Wave*. Morgan Kauffman.
332. Strommer, M., Murzek, M., & Wimmer, M. (2007). *Applying Model Transformation By-Example on Business Process Modeling Languages.* Paper presented at the Advances in Conceptual Modeling - Foundations and Applications. (ER 2007 Workshops), Auckland, New Zealand.
333. StylusStudio (2008). XML Schema Tools [Software]. Available from http://www.stylusstudio.com/xml_schema.html.
334. Sun Microsystems (1999). Java Server Pages Technology. Retrieved from http://java.sun.com/products/jsp/, November 2007.
335. Sun Microsystems (2001). Core J2EE Patterns - Data Access Object. Retrieved 20 November, 2005, from http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html.
336. Sun Microsystems. (2004). Javadoc 1.5.0. Retrieved June 2007, from http://java.sun.com/j2se/javadoc/.
337. Sun Microsytems (2007). MySQL AB [Software]. Available from http://www.mysql.com/
338. Sun Microsystems. Java Metadata Interface (JMI) Specification. (June, 2002). Retrieved, January, 2006, from: http://java.sun.com/products/jmi/index.jsp.
339. Szurszewski. J. (2003). *We Have Lift-off: The Launching Framework in Eclipse*. Eclipse Corner Article. Retrieved 5 November, 2007, from: http://www.eclipse.org/articles/Article-Launch-Framework/launch.html.
340. Tarr, P., Ossher, H., Harrison, W. & Sutton Jr, S. M. (1999). *N degrees of separation: Multi-dimensional separation of concerns.* Paper presented at the 21st International Conference on Software Engineering (ICSE 99), Los Angeles, California, United States.
341. Tikhomirov, A., & Shatali, A. (2008). *Introduction to the Graphical Modeling Framework*. Tutorial at the EclipseCON 2008. Santa Clara, California.
342. Tratt, L. (2005). Model transformations and tool integration. *Journal of Software and Systems Modeling, 4*(2), 112-122.
343. TRDCC. (2007). ModelMorf: a model transformer. Retrieved 16 October 2008, from http://www.tcs-trddc.com/ModelMorf/index.htm
344. Uhl, A. (2008). Model-Driven Development in the Enterprise. *IEEE Software, 25*(1)
345. Utz, W., & Wolfgang, K. (2003). An analysis of XML database solutions for the management of MPEG-7 media descriptions. *ACM Computer Surveys, 35*(4), 331-373.
346. Vallecillo, A. (2008). *A Journey through the Secret Life of Models.* Paper presented at the Perspectives Workshop: Model Engineering of Complex Systems (MECS), Dagstuhl, Germany.
347. Valverde, F., Valderas, P., Fons, J., & Pastor, O. (2007). *A MDA-Based Environment for Web Applications Development: From Conceptual Models to Code.* Paper presented at the 6th International Workshop on Web-Oriented Software Technologies (IWWOST), Como, Italy.

348. Van Der Sluijs, K., Houben, G. J., Broekstra, J., & Casteleyn, S. (2006). *Hera-S: Web Design using Sesame.* Paper presented at the 6th International Conference on Web Engineering (ICWE'06), Palo Alto, California (USA).
349. Van Gorp, P. (2008). *Model-Driven Development of Model Transformations.* Paper presented at the 4th International Conference on Graph Transformations ICGT 2008, Leicester, United Kingdom.
350. Van Gorp, P., Keller, A., & Janssens, D. (2009). *Transformation Language Integration Based on Profiles and Higher Order Transformations.* Paper presented at the First International Conference on Software Language Engineering, SLE 2008, Toulouse, France.
351. Vanhooff, B., Ayed, D., & Berbers, Y. (2006). *A Framework for Transformation Chain Design Processes.* Paper presented at the First European Workshop on Composition of Model Transformations - CMT 2006; European Conference on Model Driven Architecture (ECMDA-FA 2006), Bilbao, Spain.
352. Vara, J. M., Acuña, C. J., Marcos, E., & Lopez Sanz, M. (2004). Desarrollo de un Sistema de Información web: una experiencia con Oracle XMLDB. *CUORE. Círculo de Usuarios de Oracle España, 27*, 3-12.
353. Vara, J.M. De Castro, V., Marcos, E. (2005). WSDL Automatic Generation from UML Models in a MDA Framework. *International Journal of Web Services Practices, 1* (1-2), 1-12.
354. Vara, J. M., Vela, B., & Marcos, E. (2006). Oracle XML DB como repositorio integrado para herramientas CASE. Aplicación al desarrollo de MIDAS-CASE, una herramienta MDA. Paper presented at the XVI Congreso Nacional Usuarios de Oracle (CUORE).
355. Vara, J. M., Vela, B., Cavero, J. M., & Marcos, E. (2007). Model Transformation for Object-Relational Database development. SAC '07: *Proceedings of the 2007 ACM symposium on Applied computing*, 1012-1019.
356. Vara, J.M., Didonet Del Fabro, M., Joualt, F. & Bezivin, J. (2008). *Model Weaving Support for Migrating Software Artifacts from AUTOSAR 2.0 to AUTOSAR 2.1*. Int. Conf. on Embedded Real Time Software (ERTS 2008), Toulose (France), 2008.
357. Vara, J. M., De Castro, V., Didonet Del Fabro, M., & Marcos, E. (2009). Using Weaving Models to automate Model-Driven Web Engineering proposals. *International Journal of Computer Applications in Technology* (To be published).
358. Vara, J. M., Vela, B., Bollati, V., & Marcos, E. (2009). *Supporting Model-Driven Development of Object-Relational Database Schemas: a Case Study* Paper presented at the ICMT2009 - International Conference on Model Transformation, Zurich, Switzerland.
359. Varró, D., & Pataricza, A. (2003). VPM: A visual, precise and multilevel metamodelling framework for describing mathematical domains and UML (The Mathematics of Metamodelling is Metamodelling Mathematics). *Software and Systems Modeling, 2*(3), 187-210.
360. Varró, D., & Pataricza, A. (2004). *Generic and Meta-Transformations for Model Transformation Engineering.* Paper presented at the 7th International Conference on Modelling Languages and Applications - UML 2004, Lisbon, Portugal.
361. Varro, G., Schur, A., & Varro, D. (2005). *Benchmarking for Graph Transformation.* Paper presented at the IEEE Symposium on Visual Languages and Human-Centric Computing 2005, VL/HCC 2005, Dallas, USA.
362. Vdovjak, R., Frasincar, F., Houben, G. J., & Barna, P. (2003). Engineering semantic web information systems in HERA. *Journal of Web Engineering, 2*(1-2), 3-26.
363. Vela, B. (2003). MIDAS/DB: A Model-Driven Development Methodology for the structural dimension of Web Information Systems. University Rey Juan Carlos, MADRID.
364. Vela B., Marcos E. (2003). *Extending UML to represent XML Schemas.* The 15th Conference On Advanced Information Systems Engineering. CAISE’03 FORUM. Ed: J. Eder, T. Welzer. Short Paper Proceedings. Klagenfurt/Velden (Austria). 16-20 June 2003.
365. Vela, B., Acuña, C. J., & Marcos, E. (2004). A Model Driven Approach for XML Database Development. In *Conceptual Modeling - ER 2004, 23rd International Conference on Conceptual Modeling* (Vol. 3288, pp. 780-794): Springer.

366. Vela, B., Fernandez Medina, E., Marcos, E., & Piattini, M. (2006). Model driven development of secure XML databases. *ACM SIGMOD Record, 35*(3), 22-27.
367. Verner, L. BPM: The Promise and the Challenge. *Queue of ACM,* 2(4). pp. 82-91
368. Völter, M. & Kolb, B. (2006). Best practices for model-to-text transformations, *Eclipse Summit Europe 2006 - Modelling Symposium*. Esslingen, Germany.
369. Völter, M. (2006). *openArchitectureWare a flexible Open Source platform for model-driven software development*. Position paper at Eclipse Technology eXchange workshop (eTX) at ECOOP'06, Nantes, France.
370. Völter, M. (2008). MD* Best Practices. Retrieved February 20, 2008, from http://www.voelter.de.
371. Vojtisek, D., & Jézequél, J.-M. (2004). MTL and Umlaut NG - Engine and Framework for Model Transformation. *ERCIM News*(58).
372. Wadswroth, Y. *What is Participatory Action Research?* Action Research International. Recuperado de: http://www.scu.edu.au/schools/sawd/ari/ari-wadsworth.html, 2005.
373. Wagelaar, D. (2008). *Composition Techniques for Rule-Based Model Transformation Languages* Paper presented at the 1st International conference on Theory and Practice of Model Transformations (ICMT 2008), Zurich, Switzerland.
374. Warmer, J. & Kleppe, A. (2003). *The Object Constraint Language: Getting your models ready for MDA, 2nd. Edition*. Addison Wesley.
375. Watson, A. (2008). A Brief History of MDA. *UPGRADE, IX*(2), 7-12.
376. Webratio. URL, http://www.webratio.com. Last time visited: 29th of January 2007.
377. Weis, T., Ulbrich, A. & Geihs, K. (2003). Model Metamorphosis. *IEEE Software, 20*(5), 46-51.
378. Westermann, U., & Lass, W. (2003). An Analysis of XML Database Solutions for the Management of MPEG-7 Media Descriptions. *ACM Computing Surveys, 35*(4), 331-373.
379. Willink, E. D. (2008). On Challenges for a Graphical Transformation Notation and the UMLX Approach. *Electronic Notes in Theoretical Computer Science, 211*, 171-179.
380. Wimmer, M., & Kramler, G. (2006). *Bridging Grammarware and Modelware.* Paper presented at the Satellite Events at the MoDELS 2005 Conference, Montego Bay, Jamaica.
381. Winter, A., Kullbach, B. & Riediger, V. (2002). An Overview of the GXL Graph Exchange Language. Paper presented at the *Software Visualization, International Seminar*, Dagstuhl Castle, Germany.
382. W3C (1999). XML Path Language (XPath) Version 1.0. Retrieved from http://www.w3.org/TR/xpath, December 2007.
383. W3C (2003). *Web Services Description Language (WSDL) Version 1.2*. Retrieved from: http://www.w3.org/TR/wsdl12/, November, 2005.
384. W3C. (2004). *XML Extensible Markup Language (XML) 1.0 (Third Edition)*. W3C Recommendation. Bray, T., Paoli, J, Sperberg-McQu4een, C. M., Maler, E. and Yergeau F. Retrieved from: http://www.w3.org/TR/2004/REC-xml-20040204/, 2004.
385. W3C XML Schema Working Group. (2001). *XML Schema Parts 0-2:[Primer, Structures, Datatypes]*. W3C Recommendation. Retrieved from: http://www.w3.org/TR/xmlschema-0/, http://www.w3.org/TR/xmlschema-1/ y http://www.w3.org/TR/xmlschema-2/, 2001.
386. W3C (2004). *W3C Document Object Model*. Retrieved July 30, 2006, from: http://www.w3.org/DOM/.
387. W3C. (2004). *RDF/XML Syntax Specification*. Retrieved July 30, 2006, from: http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/
388. W3C. (2004). *RDF Vocabulary Description Language 1.0: RDF Schema*. Retrieved July 30, 2006, from: http://www.w3.org/TR/rdf-schema/
389. W3C. (2007). *XSL Transformations (XSLT) Version 2.0*. W3C Recommendation. Retrieved December 12, 2008, from: http://www.w3.org/TR/xslt20/
390. W3C. (2008). Definition of the XML document type declaration from Extensible Markup Language (XML) 1.0 (15th Edition). Retrieved December 12, 2008, from: http://www.w3.org/TR/REC-xml/#dt-doctype.

391. XMLmodeling.com (2008). hyperModel [Software]. Available from http://xmlmodeling.com/

Acronyms

Table of Acronyms

ACRONYM	DESCRIPTION
AMW	ATLAS Model Weaver
AST	Abstract Syntax Tree
ATL	ATLAS Transformation Language
BNF	Backus-Naur-Form
CASE	Computer Aided Software Engineering
DB	DataBase
DBMS	DataBase Management System
DOM	Document Object Model
DSL	Domain Specific Language
DTD	Document Type Definition
EMF	Eclipse Modelling Framework
EMP	Eclipse Modelling Project
GMF	Generic Modeliing Framework
GPL	General Purpose Language
GXL	Graph eXchange Language
IDE	Integrated Development Environtment
INRIA	Institut National de Recherche en Informatique et en Automatique
JET	Java Emitter Templates
JMI	Java Metadata Interface
JSP	Java Server Pages
LHS	Left Hand Side

ACRONYM	DESCRIPTION
M2DAT	MIDAS MDA Tool
M2DAT-DB	MIDAS MDA Tool for DataBases
M2M	Model to Model
M2T	Model to Text
MBSE	Model-Based Software Engineering
MDA	Model-Driven Architecture
MDE	Model-Driven Engineering
MDSD	Model-Driven Software Development
MDWE	Model-Driven Web Engineering
MOF	Meta-Object Facility
MVC	Model-View-Controller
NAC	Negative Application Condition
OCL	Object Constraint Language
OMG	Object Management Group
OR	Object-Relational
ORDB	Object-Relational DataBase
PDM	Platform Dependent Model
PIM	Platform Independent Model
PSM	Platform Specific Model
RDBMS	Relational DataBase Management System
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
RHS	Right Hand Side
SAX	Simple API for XML
SOAP	Simple Object Access Protocol

ACRONYM	**DESCRIPTION**
SQL	Structured Query Language
TGG	Triple Graph Grammars
UI	User Interface
UML	Unified Modelling Language
QVT	Query/View/Transformation
W3C	World Wide Web Concortium
WSDL	Web Services Description Language
XML	eXtensible Markup Language
XSD	XML Schema Definition