Supporting Automatic Interoperability in Model-Driven Development Processes

Giovanni Giachetti Herrera

Advisor: Dr. Óscar Pastor López

Departamento de Sistemas Informáticos y Computación

UNIVERSIDAD POLITECNICA DE VALENCIA

DSIC
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

PROS
Centro de Investigación en Métodos de Producción de Software

Doctoral Thesis

Supporting Automatic Interoperability in Model-Driven Development Processes

Giovanni Giachetti Herrera

Advisor: Oscar Pastor López

Supporting Automatic Interoperability in Model-Driven Development Processes

This report was prepared by

Giovanni Giachetti Herrera

Advisor

Oscar Pastor López, Universitat Politècnica de València

Members of the Thesis Committee:

Prof. Xavier Franch, Universitat Politècnica de Catalunya

Prof. César González-Pérez, Consejo Superior de Inv. Científicas

Prof. Juan Carlos Trujillo, Universidad de Alicante

Prof. Vicente Pelechano, Universitat Politècnica de València

Prof. Matilde Celma, Universitat Politècnica de València

Centro de Investigación en Métodos de Producción de Software (PROS)
Universitat Politècnica de València
Camino de Vera s/n, 46022 Valencia
Spain
Tel: +34-963877007 Ext. 83530
Fax: +34-963877359
Web: www.pros.upv.es

To Beatriz, Bianca, and Caterina

Acknowledgements

Probably, I would need a document larger than this thesis to express all my feelings to the people that a wish to thank. If I frame this acknowledgment section in the context of this thesis, I must say that many people have interoperated in my life to carry out this doctoral thesis by following a Friendship-Driven Development process.

Thanks Oscar for your guidance in this academic adventure. Without your unconditional support in both at professional and personal levels, this thesis would never have seen the light. You have taught me the importance of enjoy my work, and that independently of the hard or difficult of the way, with perseverance and determination is always possible to achieve the goals proposed.

Another special person that I want to thank is Pele. Pele, your honesty and dedication is an inspiration for current and future generations. You have demonstrated me that the correct way is always the best one.

Manoli, Xavi, Jaelson, and Fernanda, I really appreciate all the advices and contributions that you have made to my work. Collaborate with all of you has been a real delight.

Of course, a special place in my heart is for those unconditional friends that had given me their affection throughout these years. Ana, Vicky and Peter, for be with me from the very beginning, your love and support, especially in the hard moments, are etched in my soul. Pau, Fani, Paqui, and Carlos, I really enjoyed all the moments that we lived together. In particular, those exciting conversations combined with good food and drink. Nathalie, J-Lu, Paco, Sergio E., Ignacio your great charisma and different personalities demonstrate that people with totally opposite personalities could become in best friends.

Special mention to those friends that are physically far, but close in my heart; Gonzalo and Daniel, I really hope that in a no longer distant future we can laugh together again.

Tanja, Arthur, Marce, and Sergio S. Even though you have turned up recently in my life, I feel you as very close friends.

Also, I would thank to all my colleagues and administrative personal from the ProS research center and from the Computer Science Department. Thank you for always give me a smile in the corridors and for laughing at my bad jokes.

Finally, I want to thank to the most important woman in my life, my beloved wife Beatriz and my beautiful daughters Bianca and Caterina. Bea, when we started our adventure together, I promised to follow you to the end of the world if necessary. However, I never thought that this journey would have as a result the two most wonderful gifts in my life. Thank you girls, you three are the biggest treasure of this pirate.

Abstract

When the last few years of software development evolution are analyzed, it can be observed that the technologies involved are increasingly focused on the definition of models for the specification of the intended software products. This model-centric development schema is the main ingredient for the Model-Driven Development (MDD) paradigm.

In general terms, the MDD approaches propose the automatic generation of software products by means of the transformation of the defined models into the final program code. This transformation process is also known as the model compilation process. Thus, MDD is oriented to reducing (or even eliminating) manual programming, which is both an error-prone and time-consuming task. Hence, models become the main actors of the MDD processes: the models are the new programming code.

In this context, interoperability can be considered to be a natural trend for the future of model-driven technologies, where different developing and modeling approaches, tools, and standards can be integrated and coordinated to reduce the implementation and learning time of MDD approaches as well as to improve the quality of the final software products. However, there is a lack of approaches that provide a suitable solution to support interoperability in MDD processes. Moreover, the proposals that define an interoperability framework for MDD processes are still in a theoretical space, they are not aligned with current standards, other interoperability approaches, and existing technologies.

Thus, the main objective of this doctoral thesis is to develop an approach to achieve the interoperability in MDD processes. This interoperability approach is based on current metamodeling standards, modeling language customization mechanisms, and model-to-model transformation technologies. To achieve this objective, novel approaches have been defined to improve the integration of modeling languages, to obtain a suitable interchange of modeling information, and to perform automatic interoperability verification.

For the validation and verification of the proposed interoperability approach, empirical studies have been carried out to determine the completeness of the interoperability with regard to the modeling needs of the MDD processes involved. Also, the proposed interoperability approach has been applied to linking

UML and the *i** framework with an industrially-applied MDD approach. From these two interoperability scenarios, important feedback has been obtained to improve the approach proposed. These scenarios also show how to put in practice the results of this thesis and report interesting results for the MDD community.

Resumen

Analizando la evolución del desarrollo de software durante los últimos años, es posible observar que las tecnologías involucradas se están enfocando cada vez más en la definición de modelos para especificar los productos de software requeridos. Este esquema de desarrollo centrado en modelos es el ingrediente principal para el paradigma de Desarrollo de Software Dirigido por Modelos (DSDM).

En términos generales, las aproximaciones DSDM proponen la generación de productos de software de manera automática mediante la transformación de los modelos definidos en el código del programa final. Este proceso de transformación también es conocido como proceso de compilación de modelos. De esta manera, DSDM está orientado a reducir (o incluso eliminar) la programación manual, que es una tarea lenta y propensa a errores. Por lo tanto, los modelos se convierten en los actores principales de los procesos DSDM: los modelos son el nuevo lenguaje de programación.

En este contexto, la interoperabilidad puede ser considerada como una tendencia natural para el futuro de las tecnologías dirigidas por modelos, en donde, distintas aproximaciones de desarrollo, modelado, herramientas, y estándares pueden ser integrados y coordinados para reducir los tiempos de implementación y de aprendizaje de las aproximaciones DSDM, y consecuentemente, mejorar la calidad de los productos de software. Sin embargo, existe una carencia de aproximaciones que provean soluciones adecuadas para soportar la interoperabilidad en procesos DSDM. Además, las propuestas que definen marcos de interoperabilidad para procesos DSDM aún se encuentran a nivel teórico y no están alineadas con los estándares actuales, otras aproximaciones de interoperabilidad o tecnologías existentes.

Por este motivo, el objetivo principal de esta tesis es desarrollar una aproximación para conseguir la interoperabilidad en procesos MDD. Esta aproximación de interoperabilidad está basada en estándares actuales de metamodelado, mecanismos para la personalización de lenguajes de modelado, y tecnologías para realizar transformaciones de modelo a modelo. Para alcanzar este objetivo, se han definido propuestas innovadoras para mejorar la integración de lenguajes de modelado, obtener un adecuado intercambio de información de modelado, y verificar automáticamente la interoperabilidad.

Para validar y verificar la aproximación de interoperabilidad propuesta, se han realizado estudios empíricos que determinan la completitud de la interoperabilidad en relación a las necesidades de modelado de los procesos MDD involucrados. Además, la aproximación propuesta ha sido utilizada para conseguir la interoperabilidad de UML y del marco *i** con una propuesta DSDM de aplicación industrial. A partir de estos dos escenarios de interoperabilidad, se ha obtenido información relevante para mejorar la propuesta desarrollada. Estos escenarios también muestran cómo aplicar los resultados la tesis y proporcionan resultados interesantes para la comunidad DSDM.

Resum

Si s'analitza l'evolució del desenvolupament de programari durant els últims anys, és possible observar que les tecnologies involucrades s'enfoquen cada vegada més cap a la definició de models per especificar els productes de programari requerits. Aquest esquema de desenvolupament centrat en models és l'ingredient principal per al paradigma de Desenvolupament de Programari Dirigit per Models (DPDM).

En termes generals, les aproximacions DPDM proposen la generació de productes de programari de manera automàtica mitjançant la transformació dels models definits en el codi del programa final. Aquest procés de transformació és conegut com procés de compilació de models. D'aquesta manera, DPDM està orientat a reduir (o fins i tot eliminar) la programació manual, que és una tasca lenta i propensa a errors. Així doncs, els models es converteixen en els actors principals del processos DPDM: els models són el nou llenguatge de programació.

En aquest context, la interoperabilitat pot ser considerada com una tendència natural per al futur de les tecnologies dirigides per models, on distintes aproximacions de desenvolupament, modelat, ferramentes, i estàndards, poden ser integrats i coordinats per tal de reduir els temps d'implementació i d'aprenentatge de les aproximacions DPDM i, conseqüentment, millorar la qualitat dels productes de programari. Tanmateix, existeix una mancança d'aproximacions que proveïsquen solucions adequades per donar suport a la interoperabilitat dels processos DPDM. A més, les propostes que defineixen marcs d'interoperabilitat per a processos DPDM encara es troben a nivell teòric i no estan alineats amb els estàndards actuals, altres aproximacions d'interoperabilitat o tecnologies existents.

Per aquest motiu, l'objectiu principal d'aquesta tesi és desenvolupar una aproximació per aconseguir la interoperabilitat en processos DPDM. Aquesta aproximació d'interoperabilitat està basada en estàndards actuals de metamodel, mecanismes per a la personalització de llenguatges de modelat, i tecnologies per a realitzar transformacions de model a model. Per aconseguir aquest objectiu, s'han definit propostes innovadores per millorar la integració de llenguatges de modelat, obtenir un intercanvi d'informació de modelat adequat, i verificar automàticament la interoperabilitat.

Per validar i verificar l'aproximació d'interoperabilitat proposada, s'han realitzat estudis empírics que determinen la completesa de la interoperabilitat en relació a les necessitats de modelat dels processos DPDM involucrats. A més, l'aproximació proposada ha estat utilitzada per aconseguir la interoperabilitat de UML i del marc i* amb una proposta DPDM d'aplicació industrial. A partir d'aquests dos escenaris d'interoperabilitat, s'ha obtingut informació rellevant per a millorar la proposta presentada. Aquests escenaris també mostren com aplicar els resultats de la tesi i proporcionen resultats interessants per a la comunitat DPDM.

Contents

List of Figures

List of Tables

Chapter I.
Introduction

When the last few years of software development evolution are analyzed, it can be observed that the technologies involved are increasingly focused on the definition of models for the specification of the intended software products. This model-centric development schema is the main ingredient for the Model-Driven Development (MDD) paradigm.

In general terms, the MDD approaches propose the automatic generation of software products by means of the transformation of the defined models into the final program code. This transformation process is also known as the model compilation process. Thus, MDD is oriented to reducing (or even eliminating) manual programming, which is both an error-prone and time-consuming task. Hence, models become the main actors of the MDD processes: the models are the new programming code.

In this context, interoperability can be considered to be a natural trend for the future of model-driven technologies, where different developing and modeling approaches can be integrated and coordinated to reduce the implementation and learning time of MDD approaches as well as to improve the quality of the final software products. Thus, we have developed this doctoral thesis with the aim of providing an approach that can be used as a reference to achieve interoperability in MDD processes.

One of the most important concerns when elaborating a Model-Driven Development (MDD) [1, 2] solution is the specification of a modeling language that allows the required software products to be represented at the conceptual level without ambiguity. Among the different choices that exist for the definition of an adequate modeling language, there are two alternatives that appear to be the most suitable. The first of these is the creation of a proprietary *Domain-Specific Modeling Language* (DSML) [3, 4] for the MDD approach. The second alternative is the customization of existing modeling languages, which are normally supported by a standard or are used as a de-facto standard in specific domains. Thus, generally speaking, these modeling languages are well-known and mature modeling approaches, which have been empirically validated. Existing modeling approaches can be multi-purpose modeling languages such as UML [5] or, by contrast, modeling languages related to a specific domain, such as *i** [6, 7] for requirement modeling.

Both the customization of existing modeling languages and the specification of proprietary DSMLs are suitable modeling alternatives that provide interesting benefits for the application of MDD approaches. In practice, these two modeling alternatives are viewed as opposite solutions [8], which cannot be integrated into a common MDD process. This situation is principally caused by the lack of an appropriate solution that indicates how to interoperate the modeling information produced by different modeling approaches. We believe that by applying an appropriate interoperability solution, the existing modeling approaches and the proprietary DSMLs can be used as complementary alternatives to perform the modeling tasks that are involved in a MDD process.

A modeling framework that considers the interoperability of different modeling approaches is a suitable alternative for supporting real software production processes where different roles must collaborate at different development stages [9] (e.g. project managers, requirement analysts, system designers, and programmers). These roles may use different developing tools and modeling technologies, which can be supported by existing modeling languages or by DSMLs that are developed in the context of the MDD process. Therefore, the modeling approaches involved must interoperate with each other in order to link the different development stages. This situation is clearly observed in MDD approaches that are based on the Model-Driven Architecture (MDA) [10, 11] proposed by OMG, which is one of the most widely used MDD implementation strategies. MDA proposes the consecutive transformation of models from higher abstraction levels to lower abstraction levels, until the final software product is achieved. Figure 1 shows the MDA schema.

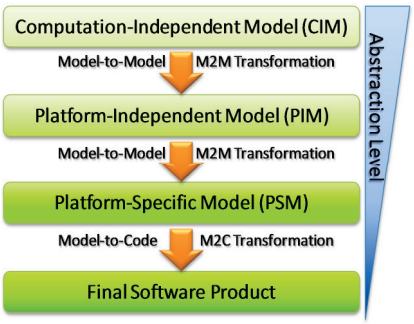

Figure 1. The MDA schema

MDA suggests the use of UML to perform the modeling tasks that are related to the CIM, PIM and PSM levels. However, other modeling approaches can also be used in order to apply the most suitable solution for each abstraction level. Additionally, modeling approaches can be combined inside of each abstraction level to obtain a precise conceptual representation from different perspectives, for instance, the dynamic, structural, and presentation perspectives [12].

The interoperability of different modeling approaches is not only suitable for MDD approaches that are based on MDA. It can be generalized for any MDD approach that wants to achieve a sound Model-Driven Engineering (MDE) [13] process, where different abstraction levels and modeling perspectives must be coordinated. This situation is discussed in works such as [14, 15].

The main objective of this doctoral thesis is to develop a suitable approach for the interoperability of existing modeling approaches, standards, and tools with specific MDD modeling approaches and technologies, such as proprietary DSMLs and model compilers. This interoperability approach is based on current metamodeling standards, modeling language customization mechanisms, and model-to-model transformation technologies.

We have verified the proposed interoperability approach by linking two relevant modeling approaches with an industrially-applied MDD process. The modeling approaches are: 1) *UML*, which is a well-known general-purpose modeling language, and 2) the *i* framework* [6], which is a goal-oriented requirement modeling approach. The MDD approach involved is OO-Method [16], which has a proprietary DSML for representing the different constructs that comprise its conceptual model [12].

The linking of UML and *i** with the OO-Method MDD approach presents two interesting interoperability scenarios, which provide relevant benefits for MDD approaches. The development and execution of these interoperability scenarios are involved in the iterations that have been performed to develop, fix, improve, and finally obtain the interoperability approach proposed in this thesis. Appendix II provides additional details about the thesis development process.

The rest of this introduction chapter is organized as follows: Section 1.1 provides additional motivation for the work presented. Section 1.2 states the problems that exist in the context of MDD development, which drive the objectives of this thesis. Section 1.3 details the objectives to be achieved. Section 1.4 presents the general context in which this thesis is developed. Finally, Section 1.5 presents the structure of the whole document

1.1. Motivation

The motivation behind the work performed in this thesis lies in the benefits that a multiple-modeling schema can provide for the implementation and application of a specific MDD approach. In particular, we consider the benefits related to the definition of specific DSMLs, and the adoption of existing modeling languages.

On one hand, a specific DSML provides a precise characterization of the conceptual constructs that are required for the definition of the models that are involved in a MDD process. Furthermore, the number of conceptual constructs required by a MDD proposal is generally smaller than the number of conceptual constructs present in more general modeling approaches (such as UML). This facilitates the model compilation process and the implementation of specific MDD tools with improved modeling features. However, it is important to note that the implementation and maintenance of these specific tools involve extra effort when putting an MDD approach into practice [17].

On the other hand, an existing modeling approach can be customized to represent the particular modeling needs of a MDD approach. The use of existing modeling approaches has a very powerful argumentation, which is: do not reinvent the wheel and take advantage of existing tools, technologies, theories, user experience, etc., to put into practice a particular MDD approach [18]. This is especially true for standard (or de-facto standard) modeling approaches because languages of this kind have more users and related technologies.

It is clear that the interoperability of proprietary DSMLs and customized modeling languages provides interesting benefits for the application of MDD approaches. However, there are certain issues that must be tackled to achieve this interoperability. The most important issues are presented in the next section.

1.2. Problem Statement

Nowadays, since the lack of a suitable solution to link existing modeling languages with a specific MDD process, several MDD proposals have defined *Domain-Specific Modeling Languages* (DSMLs) [3, 4] to represent their modeling needs. This has produced the proliferation of several modeling approaches with their respective notation and modeling tools. As a consequence, it is very difficult to interchange knowledge among the different approaches that exist in the MDD community. It also difficult to compare the existing approaches to contrast their pros and to select an appropriate MDD solution that is aligned with the needs of a specific project or organization.

In general terms, many of the modeling aspects considered by modeling approaches related to a specific domain can be generalized. This situation is observed in MDD proposals with a class model specification. In this case, most of the concepts of the class model (such as classes, associations, attributes, services, etc.) can be represented with more general modeling approaches, for instance, with the UML class model. Thus, it would be possible to customize the UML class model to represent the concepts involved in the specific MDD approach. This same idea can be applied to other modeling languages such as the *i** framework for goal-oriented requirement modeling, where MDD proposals can customize this modeling language to represent their requirement modeling needs.

Current modeling approaches are based on the definition of metamodels to specify the abstract syntax of the required conceptual constructs, relationships, and validation rules. Thus, the customization of modeling languages can be performed by extending the involved metamodels. In an interoperability context, this customization must be performed with a mechanism that does not change the target metamodels. This constraint is oriented to ensuring compatibility with existing standards and implemented technologies.

Currently, there is not a standardized process that states how to define metamodel extensions for the customization of a modeling approach. For this

reason, the customization of a modeling language is usually elaborated in a straightforward way without a well-defined process. Thus, it is common for customized languages not to be aligned with the standards [17]. In addition, the manual and intuitive definition of the required metamodel extensions for modeling language customization is an error-prone and time-consuming task [19]. These two risk factors (time and error) must be avoided, especially in a MDD industrial context, where time costs money and mistakes in implementation directly impact on customer satisfaction.

Even though the use of standardized modeling languages could improve the application of specific MDD approaches by taking advantage of user experience and existing technology, the construction of specific MDD tools such as model compilers is usually harder from standardized modeling languages than from specific DSMLs. This is mainly due to the extra size and complexity of the standard modeling languages in relation to the specific DSMLs. The Proprietary DSMLs only provide the specific constructs and properties required for an MDD approach, while a standard language provides a larger number of constructs and properties for a more general application.

In summary, a sound interoperability framework for MDD process must support the coordination of different modeling approaches that participate at different abstraction levels (such as requirement models, business process models, and different kinds of software specification models). However, a standard solution to achieve this interoperability has not yet been defined. This can be observed in proposals oriented to goal-oriented modeling, such as the *i** framework for requirement modeling and business analysis. In the *i** context, ther is no well-defined mechanism to transform *i** models into the corresponding software models [20, 21], at least not by means of an automatic model-to-model transformation process.

Certain proposals have defined mechanisms to confront the automatic transformation of models among different modeling languages, such as [22], which is based on the automatic definition of metamodel mappings. However, these kinds of approaches do not manage all of the differences (heterogeneities) that may exist among the modeling languages involved, consequently, relevant modeling information during the transformation process can be lost.

According to the different elements considered in this chapter, we can conclude that the main problem to obtain a suitable multi-modeling framework for MDD approaches is the definition of an appropriate interoperability approach for MDD processes. To confront this problem, more specific issues must be

considered, such as the integration of modeling languages and the appropriate interchange of models. Thus, the objectives that we want to achieve in this thesis are guided by this main problem and the specific issues related. These objectives are detailed in the next section.

1.3. Objectives

In order to tackle the issues presented in the problem statement section, we have defined the main goal of this thesis as follows:

The Development of a Suitable Approach for the Interoperability of Different Modeling Approaches in a Common MDD Process.

We consider that the required interoperability can be obtained through the integration and customization of the modeling approach together with a sound model-interchange mechanism that prevent the loss of information during the interchange of modeling information. Furthermore, a suitable model-based interoperability approach must provide automation facilities to reduce the errors introduced by manual definition of the required customizations or by the manual transformation of the models involved.

The interchange of information among the integrated modeling languages is essential to achieve model interoperability so that an appropriate conceptual representation at different abstraction levels can be obtained. Also, the model-to-model transformations for going from an upper abstraction level to a lower abstraction level can be automated. In addition, to facilitate the application of the interoperability approach, it must be based on existing standards and technologies for metamodeling, modeling language customization, and model transformations.

Thus, the main goal of this thesis is supported by the following specific objectives:

1) The creation of a well-defined process for the customization of modeling languages with the modeling needs of specific MDD approaches. This process must have the following features:

    a. It must allow the automatic generation of metamodels extensions to integrate the abstract syntax of the reference MDD approaches into the modeling approaches involved.

b. The metamodel extensions must not modify the specification of the target metamodel to assure compatibility with existing technology and standards.

2) The definition of a mechanism for automatic interchange of information among models that are defined with customized modeling languages and models that are defined with specific DSMLs. This interchange mechanism must have the following features:

   a. The interchange of models must be performed by means of model-to-model transformations that are based on the metamodels of the modeling languages involved.

   b. The interchange of models must prevent the loss of information during the transformation process.

3) The application of the proposed interoperability approach to different interoperability scenarios, which are related to different stages of a MDD process. This third objective is oriented to perform the following tasks:

   a. The improvement of the proposed interoperability approach with the results obtained from the development and execution of the different interoperability scenarios.

   b. The presentation and exemplification of the use of the proposed interoperability approach by its application in two interoperability scenarios that are relevant for MDD approaches.

   The interoperability scenarios that are considered in this third objective are related to the following development stages:

   a. The analysis level. This stage is oriented to perform the requirement elicitation in a MDD process.

   b. The design level. This stage is oriented to represent the different views of the intended software products, such as structural, behavioral, and presentation views.

1.4. Thesis Development Context

This doctoral thesis has been developed within the ProS Research Center [23] from the Universidad Politécnica de Valencia [24]. The work performed to achieve the proposed objectives is directly related to the knowledge and experience obtained from the execution of different research projects, which involve both industrial and academic efforts. These projects are the following:

1. CONCOM: This Project is oriented to the development of model compilers for the generation of different kinds of software products in the context of the OO-Method approach. This project has been financed by CARE Technologies. It was developed from 2006 to 2008.
2. SESAMO: This project is oriented to the construction of web services from models by following the model-driven development philosophy. This is a CICYT project with reference TIN2007-62894. It was developed from 2008 to 2010.
3. ORCA: This project is oriented to construct and coordinate different methods for software development in a model-driven context. The different methods are put into practice according to a specific framework that assures the quality of the generated software products as well as the quality of the methods applied. This project is financed by the Generalitat Valenciana [25] (reference PROMETEO/2009/015). It has been under development since 2009 and will continue until 2012.

Furthermore, we are starting a new project that is aligned with the results obtained in this thesis. This project, called ProsREQ [26], is not only related to interoperating different requirement modeling approaches and MDD processes to improve the generation of Web services, but also to automating the generation of verification mechanisms for the generated software products. ProsREQ is supported by MICINN [27] under reference TIN2010-18011. It is being developed by the ProS Research Center in collaboration with the GESSI research group [28] from the Universidad Politécnica de Cataluña. The ProsREQ project will be developed from 2011 to 2013.

1.5. Thesis Structure

In addition to the introduction chapter, this doctoral thesis is comprised of ten more chapters, which are organized as follows:

Chapter II – Background: This chapter introduces the main concepts that are involved in the development of this thesis. The approaches that are related to the interoperability scenarios to improve and evaluate the proposed interoperability approach are also presented. These correspond to UML, the *i** framework, and the OO-Method MDD approach.

Chapter III – Related Work: This chapter presents a systematic review of the current approaches related to model-based interoperability. This systematic review is centered on analyzing those interoperability approaches that provide relevant features for MDD processes.

Chapter IV – Achieving the MDD Interoperability: This chapter presents the aspects that are considered in order to obtain a specific process to support MDD interoperability, which is based on a specific MDD interoperability model. The steps that comprise the proposed interoperability process are also introduced.

Chapter V – The Integration Metamodel: This chapter presents a specific DSML metamodel, called Integration Metamodel, which is defined to perform a correct customization of a modeling language. The Integration Metamodel is used to automatically obtain a set of lightweight metamodel extensions that are implemented in a UML profile. As a result, a modeling language customized with the modeling needs of a specific MDD approach is obtained in accordance with the OMG modeling standards.

Chapter VI – Automatic UML Profile Generation: This chapter presents a process that is defined to integrate a particular DSML into a target modeling language through the automatic generation of a UML profile. This process facilitates the correct use of existing modeling languages in a proprietary MDD context and provides a solution that takes advantage of the benefits of standard modeling languages and proprietary DSMLs.

Chapter VII – Generation of Model Interchange Mechanisms: This chapter presents a proposal that generates mechanisms that perform model-to-model transformations to automatically interchange information among modeling approaches to must interoperate. The generation of the transformation mechanisms is performed by means of the different artifacts generated from the application of the proposed MDD interoperability process.

Chapter VIII – Linking UML and MDD Approaches: This chapter presents an interoperability scenario that is focused on the application of UML models in MDD processes, specifically, the application of the constructs related to the UML association in the OO-Method development process. This chapter also shows how the results obtained allow the generation of software products from UML models through the industrial model compiler related to the OO-Method approach.

Chapter IX – Linking Goal-Oriented Modeling and Model-Driven Development: This chapter presents a second interoperability scenario where the interoperability approach proposed in this thesis is used to automatically link GORE models and MDD processes. This scenario has been elaborated by considering the experience obtained from linking the *i** framework with the industrial implementation of the OO-Method approach.

Chapter X – Automatic Verification of Models for MDD Interoperability: This chapter presents an approach to guarantee correct MDD interoperability by means of the integration of specific verification mechanisms into the involved modeling language. This approach is based on a specific process for the definition and implementation of verification measures, which also puts into practice the interoperability approach proposed in this thesis. The results obtained from the application of this verification approach are empirically validated.

Chapter XI – Conclusions: This chapter presents the contribution obtained from the work performed in this doctoral thesis, future research lines related to the work performed, and the publications generated during the thesis development.

Chapter II.
Background

This chapter presents the basis that are necessary to understand the development and results of the interoperability approach presented in thesis. In this interoperability approach, the customization of metamodels to perform the integration of the involved modeling languages is a key issue. This integration can be considered as the glue to perform the interoperability of different modeling approaches and MDD processes by means of the appropriate interchange of modeling information. Thus, the main alternatives that exist for modeling languages customization are presented by indicating their advantages and disadvantages. Also, the modeling frameworks and MDD approaches that are involved in the interoperability scenarios are presented, which correspond to UML, the i framework, and the OO-Method MDD approach.*

The MDD proposals require modeling languages for the definition of conceptual models that represent in a complete and unambiguous way the software products to be developed. According to the IEEE Standard Glossary of Software Engineering Terminology [29] a software product corresponds to *Computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system.*

The application of existing modeling languages for the definition of conceptual models provides different advantages for MDD proposals [30], such as reduction in learning curve, reduction in implantation costs, reuse of existing technologies and modeling tools, etc. These benefits are more evident when using standard modeling languages. However, conceptual constructs of standard modeling languages usually need additional modeling information to represent the conceptual models of specific MDD approaches with all the required precision [21, 31, 32]. Thus, customization of the existing modeling languages must be performed to introduce this additional information. In this thesis, the customization of modeling languages is the Rosetta stone to obtain an appropriate MDD interoperability.

The rest of this chapter is organized as follows: Section 1.1 introduces the main modeling language customization mechanisms. Section 1.1 presents the OO-Method approach. Finally, Section 1.1 presents the *i** modeling framework.

2.1. Modeling Language Customization

In order to use existing modeling languages for the correct representation of the constructs that are related to MDD approaches, it is necessary to customize the modeling language with the information that MDD approaches require. In this case, are only considered those modeling aspects that the original definition of the target modeling languages does not provide. In other words, existing modeling languages are customized to use them as proprietary DSMLs.

There are different mechanisms for customization of modeling languages [33]. In general terms, these customization mechanisms are based on the definition of extensions over the metamodels that describe the abstract syntax of the modeling languages. The metamodel extensions not necessarily represent additional modeling capabilities. These extensions can be constraints to manage the definition of models, for instance: to reduce the set of possible values related to a property, or to manage the definition of an association between two classes.

Among the different metamodel extension mechanisms that exist [33], there are two kind that have more relevance: the *lightweight* extension mechanisms and the *heavyweight* extension mechanisms. These two extension mechanisms are detailed below.

2.1.1. Lightweight Extension Mechanisms

The lightweight extensions are denominated 'light' because these extensions do not change the reference metamodel (the metamodel of the modeling language to be customized). Thus, lightweight extensions only add constraints and new elements to the target metamodel, but they not change the already defined elements. The Object Management Group (OMG) [34] has defined a standard for the definition of lightweight extensions, which is called UML profile.

The UML profile is part of the UML specification and it is defined in the *UML Infrastructure* [35]. It indicates the mechanisms used to adapt existing MOF-based metamodels to specific platforms, domains, business objects, or software process modeling. Since this extension mechanism is a part of the UML standard, it can be supported by UML tools. This feature is one of the main advantages of the UML profile over other customization mechanisms [33], which are not directly supported by modeling tools.

A UML profile is represented as a UML package that is stereotyped with the tag <<profile>>. It has three main constructs for the definition of the required extensions: stereotypes, tagged values, and OCL rules:

- The stereotype is the central construct for the specification of a UML profile. It is a special kind of UML class (specialization of the metaclass *Class* from the UML metamodel). Therefore, the semantics and notation of a stereotype is very similar to a UML class. The stereotypes are identified by a unique name, and represent the set of the extensions that are applied over the classes of the extended metamodel. The extended classes are identified by means of extensions relationships that go from the stereotypes to the metaclasses that they extend. The tagged values and the OCL rules are used to characterize the extensions related to a stereotype.
- A tagged value is a property (specialization of the UML metaclass *Property*) that is owned by a stereotype. A tagged value represents a new property that is added to the metaclass extended by the stereotype that owns the tagged value. According to the lasts UML specifications (UML 2.1.1 [36] and above), the type of the tagged values can be specified from other classes or

stereotypes. Therefore, the tagged values can be used for the definition of new attributes as well as for the definition of new associations among the metaclasses of the extended metamodel.

- The OCL rules are defined by means of the Object Constraint Language [37]. Each OCL rule is related to a specific stereotype and are used to control the interaction among the different conceptual constructs (extended metaclasses). Even though the name OCL makes reference to the definition of constraints, the last OCL specification can also be used as a query language and as a language for the specification of functions and operations.

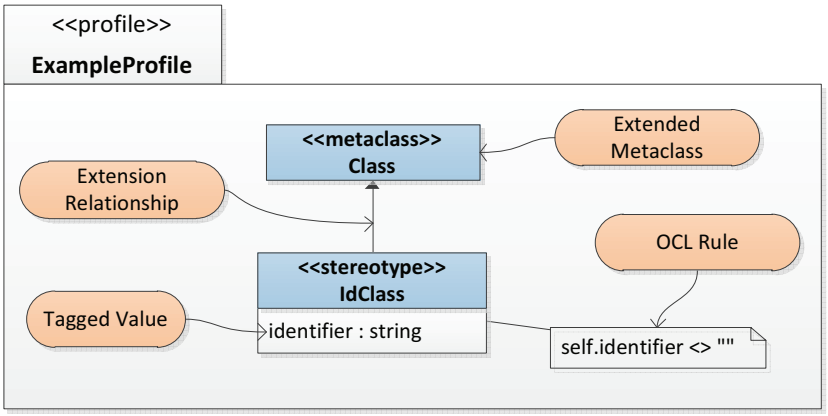

Figure 2. UML profile example

Figure 2 shows a brief example of a UML profile. This UML profile has the stereotype *IdClass* that extends the metaclass *Class* with an identifier, which is specified by means of the tagged value *identifier*. The stereotype *IdClass* also has an OCL rule that indicates that the assignment of a value for the identifier is mandatory.

From lasts UML versions, (UML 2.0 and above) the UML specification is perfectly aligned to the metamodeling standard defined by OMG, which is the Meta-Object Facility (MOF) [38]. This means that the extension capabilities of UML profiles can also be applied to any MOF-Based metamodel, such as the UML metamodel is. Thus, the UML profile could be used to extend any modeling language that uses a MOF metamodel for the specification of its abstract syntax, which is suitable to achieve the MDD interoperability proposed in this thesis.

The standardization of UML profiles (in the UML specification [39]) facilitates the use of this extension mechanism for the integration of multiple modeling languages, which is one the objectives of this thesis. The standardization prevents compatibility problems in the definition of extensions related to different modeling approaches that must interoperate. In addition, UML profile has standardized interchange support defined in the XMI format [40]. The standardization of the XMI definition for the UML profiles prevents compatibility problems to transfer modeling information among model-management tools.

Since UML profile is a lightweight extension mechanism, the defined extensions cannot change the target metamodel. Thus, UML profiles can only define extensions derived from the constructs that already exist in the extended modeling approach. Moreover, the defined extensions cannot change the properties of the original modeling constructs. These limitations of UML profiles can be considered an advantage and a disadvantage at the same time.

On one hand, these limitations are an advantage because if the original specification of the target metamodel is not changed by the defined extensions, the technologies based on the original modeling language can be also applied over models defined with the extended modeling language.

On the other hand, the limitations of UML profiles are a disadvantage since it may difficult to represent certain modeling needs of MDD approaches from existing modeling constructs.

2.1.2. Heavyweight Extension Mechanisms

The heavyweight extensions, also known as first level extensions, allow the extension of modeling approaches through the inheritance of meta-types from the referenced meta-model. Thus, these metamodel extensions can modify the target metamodel by redefining properties of the constructs that already exist in the modeling language. In addition, at difference of lightweight extensions, heavyweight extensions can add new conceptual constructs to the extended modeling languages from scratch.

For the definition of heavyweight extensions, it is necessary to select the constructs to be extended from the target modeling language. Next, the selected construct are merged in a new package where the extension are defined, including the definition of new conceptual constructs.

The definition of heavyweight extensions is based on the features provided by the MOF standard [38]. In this metamodeling standard, the package merging is a very relevant feature that is even used for the specification of the UML metamodel, the UML superstructure [41]. In addition, there are other interesting MOF features relevant for the definition of heavyweight extensions, one of these, the property redefinition, is the feature that allows the modification of properties of the constructs that already exist in the extended modeling language. Figure shows a property redefinition example, in this example the association related to the member ends of a generic relationship is redefined by the association *memberEnd* of the construct related to binary associations between classes. In this redefinition example, the type and cardinality of the original member end are changed.

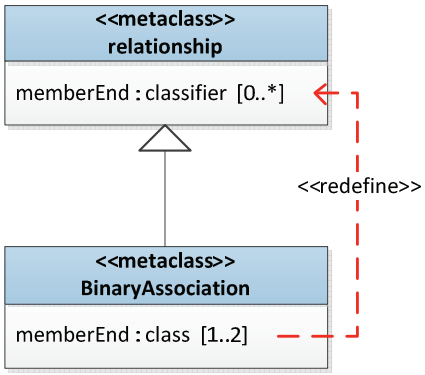

Figure 3. Property redefinition example

The main advantage of heavyweight extensions is that they have more flexibility than lightweight extensions, providing support to represent those modeling features that lightweight extensions (considering UML profile features) cannot represent.

The main disadvantage of heavyweight extensions is that they change the target metamodel, thus the result is a new metamodel that differs from the original specification of the extended modeling language. Therefore, the extended metamodel is no-longer compatible with technologies that are based on the original modeling language. This situation is relevant because the reuse of existing technologies is one of the main advantages of the interoperability among modeling languages. In addition, in contrast to lightweight extensions, heavyweight extensions are not supported by standards for their specification and interchange, which difficult the correct application of this extension mechanism

in heterogeneous modeling frameworks for validation of defined extensions and the interchange of extended models.

The report performed by James Bruck and Kenn Hussey [33] shows in more detail the different extension mechanisms for metamodels customization. Even though this work is centered on the specification of extensions for UML, the presented extension mechanisms and analyzed features are applicable to any standard modeling language that uses a MOF metamodel. In this work, it is also explained why lightweight extension mechanism is the most suitable for extend modeling languages, leaving the heavyweight extensions in a second place. Thus, heavyweight extensions are usable only when lightweight extensions do not provide the required flexibility for the representation of the needed modeling features. The work presented by Staron and Wohlin [30] is an interesting case study that shows the advantage of using lightweight extensions (trough UML profiles) instead of heavyweight extensions.

We can conclude that the lightweight extensions are the most suitable strategy for the customization of modeling languages in order to obtain appropriate support for interoperability among different modeling approaches. Lightweight extensions do not change the metamodels of the involved modeling languages, and, hence, current MDD technologies (such as model compilers) can be used over models defined with the extended modeling languages. We have chosen the UML profile extension mechanism for the generation of the metamodel extensions that are required to support the objectives of this thesis.

2.1.3. Definition of UML Profiles

Currently, there are many proposals related to specific domains that have used UML profiles to implement their modeling needs [42]. Some of these proposal have been adopted by OMG and can be found in the UML profiles specification catalog presented in [43]. It is also possible to find UML profiles oriented to describe the conceptual models required by model compilers involved in MDD processes, such as *WebML* [44] and *OO-Method* [45] proposals.

In despite that UML profiles are longer used for the customization of UML, OMG has not defined a standardized process focused on the correct implementation of UML profiles yet. This situation is presented in the article of France, *et al.* [31].

In the literature related to the definition of UML profiles, two main working schemas can be observed: 1) the definition of a UML profile from scratch; and 2)

the definition of a UML profile starting from a *DSML Metamodel* [17], which is the metamodel that describes the conceptual constructs required by a MDD approach.

The first working schema implies the direct definition of the UML profile extensions in a manual and intuitive way according to the knowledge and criteria that the UML profile designer has. An example of this implementation schema is the *SysML* UML profile [46] [47]. This intuitive method for defining UML profiles presents a high complexity degree. It difficult the verification of the required extensions in relation to the modeling needs, and their alignment with the UML profile standard. Also, this working schema is very time consuming and susceptible to the introduction of human errors.

The second working schema considers a more structured and formal process. This schema is centered on the definition of a metamodel that describes the conceptual constructs required by a specific modeling approach. In the context of this thesis, this metamodel describes the abstract syntax that supports the semantics [48] of the modeling language of the MDD approach; i.e., this is the DSML metamodel of the MDD approach. From the defined metamodel, a mapping (or model weaving [49]) to the modeling language to be extended is defined. This mapping is performed to identify the correspondences (equivalences) among the conceptual constructs of the MDD approach and the constructs of the target modeling language. Thus, it is possible to identify the constructs of the target modeling languages that are relevant for the representation of the MDD constructs. This mapping also allows the identification of those MDD constructs that are not present in the target modeling language and must be defined as metamodel extensions.

For the interoperability approach presented in this thesis, the second working schema has been selected since it provides a methodological solution that facilitates the correct definition of required modeling needs, and provides more automation possibilities. Additional benefits of using a metamodel definition for the automatic UML profile generation are the following:

- The definition of a metamodel provides a formal and precise abstract syntax related to the conceptual constructs of the MDD approach. This allows the specification of validation mechanism to assure the correct definition of the syntax for the required constructs. In addition, the specification of the abstract syntax of the target modeling language in a metamodel also facilitates the identification of equivalences among the constructs of the MDD approach and the constructs that the target modeling language

provides. This facilitates the implementation of mechanism for the automatic generation of metamodel extensions that integrate into the target modeling language the modeling needs of the MDD approach.

- In general terms, the metamodel of a proprietary DSML provides a less number of constructs than an existing (standard) modeling language that is not defined for the MDD approach. In addition, the definition of a specific metamodel perfectly fits with the application domain of the MDD approach. Both, the reduced number of constructs and the closeness to the application domain provokes that the definition of the metamodel related to the MDD approach be more simple and intuitive than the direct definition of extensions over the target modeling language. In addition, implementation of specific MDD tools (such as model compilers) is easier from the proprietary DSML metamodel than from the customized modeling language.

- There is a lot of literature related to the definition of metamodels, and there also exist tools for the correct specification of metamodels. However, the documentation related to the correct specification of for UML profiles is very limited, and the existing UML tools provide little or none aid for the correct implementation of metamodel extensions. Additionally, most of the lightweight extension papers are oriented to define UML profiles for UML while, in the context of this thesis, the extension capabilities the UML profiles are considered for the customization of any MOF-based metamodel.

- The formalization of the required abstract syntax in a metamodel helps to determinate if the modeling needs of the MDD approach can be integrated in the target modeling language according to the extension capabilities of the UML profiles. Additionally, if the integration is not possible by using UML profiles, then the DSML metamodel can be also used to implement the required metamodel extensions with another extension mechanism, such as heavyweight extensions, or for implementation of specific model editors by using tools such as Eclipse GMF [50].

One of the first works related to the elaboration of a UML profile form a metamodel specification is the work presented by Fuentes-Fernández, *et al.* in [51]. In this article, the metamodel is called *Domain Metamodel*, and some basic guidelines for the generation of the UML profile are proposed. It proposes the use of MOF (Meta-Object Facility) [38, 52] for defining the metamodel of the proprietary DSML. MOF is a standard metamodeling language that has tools support and a standardized interchange format, which facilitates the implementation of technologies based on this standard. The use of MOF for the

specification of the participant modeling languages also facilitates the identification of structural similarities (equivalences) among conceptual constructs.

This paper proposes basic guidelines for the inference of metamodel extensions (stereotypes, tagged values, and OCL constraints) as well as for the identification of the metaclasses that must be extended from the target metamodel. These guidelines provide a first approach about how to automate the UML profile generation. However, the guidelines proposed for the UML profile construction are very basic and they not take advantage of all the modeling features already present in the target modeling language for the representation of the necessary MDD constructs. For instance, this approach proposes the definition of all attributes present in a metaclass of the proprietary DSML as tagged values in the corresponding metaclass of the target modeling language. This not considers those existing attributes of the target metaclass that represent a semantics that is equivalent to the attributes of the metaclass from the proprietary DSML.

A relevant point that is missed in this paper is to indicate how the metamodel of the proprietary DSML can be defined. In particular, to indicate how to obtain a metamodel that allows the correct integration (definition of the required metamodel extensions) to the target modeling language.

In [17], Selic presents a systematic approach (updated in [53]) to define UML profiles starting from a DSML metamodel, which is also called *Domain Model*. This work shows a set of criteria that must be considered at the moment of defining the corresponding metamodel. These criteria are oriented to provide a correct integration to the target modeling language (UML in this article). In addition, Selic also proposes a list of elements that must be considered for mapping the defined metamodel into a UML profile.

The Selic's work provide interesting information about the new features introduced in the UML profile specification from UML 2.x specification [41]. At this point, important aspects of the UML profile extensions capabilities are presented, such as the capability of import model libraries, and the possibility of introduce new association in the extended metamodels by means of object-valued property definition

This is one of the former works that present guidelines for an appropriate DSML metamodel definition, which supports the appropriate specification of the required UML profile. Thus, from the defined DSML metamodel, it is possible

to identify equivalences with the metamodel of the target modeling language, which facilitates the identification of the elements to be extended and the extensions that must be defined. This is also a relevant point to be considered to obtain an automated UML profile generation.

However, the guidelines proposed for the UML profile construction are very simple, which impede an effective implementation of automatic UML profile generation solution. Additionally, it not explain how to solve the differences between participant metamodels (source DSML and target modeling language) that prevent the correct identification of the elements to be extended and definition of required modeling extensions.

Going deeper in the line of automating the UML profile generation from a DSML metamodel, there are two remarkable articles, these are: Lagarde, *et al.* [54] and Wimmer, *et al.* [19].

The Lagarde, *et al.* article proposes the identification of equivalences between a source metamodel (DSML metamodel) and a target metamodel (UML metamodel) through the definition of an initial skeleton of the required UML profile. Later, the defined skeleton is refined to obtain the final UML profile through patterns identification. A positive aspect of this work is the definition of an appropriate input for the mapping of equivalence between metamodels and later generation of the corresponding UML profile. This input is the UML profile skeleton, which demands additional knowledge about UML profile definition. However, this approach is mainly centered on binary associations between classes, the rest of metamodeling constructs are not considered, and, hence, it is not possible to perform a complete generation of the UML profile from the DSML metamodel. The application of the Lagarde, *et al.* proposal is presented in [55] and [56].

The Wimmer, *et al.* article proposes a semiautomatic approach for the integration of DSML and UML. In this proposal a new specific language for the definition of a weaving model (mapping model) between the metamodels involved is defined. This model weaving definition is oriented to identify the equivalences among the different constructs of the DSML metamodel and the UML metamodel. In this paper, the EMOF specification [38] is used for the definition of the reference DSML metamodel, in particular, the eclipse EMF implementation [57]. Both, the EMOF metamodel and the mapping model, are used as input for a model-to-model transformation tool that is based on ATL. This clearly shows how the automatic generation of the corresponding UML profile can be implemented by using model-to-model transformation technologies,

such as *ATL* [58] or *QVT* [59], which are based on mappings defined at metamodel level.

However, this proposal does not support all possible mappings that can be defined between the participant metamodels. It only supports certain one-to-many (1:M) mappings (one element of the DSML metamodel is mapped to many elements of the UML metamodel), and it not supports M:M mappings. This limitation is relevant for the effective application of this proposal into real MDD approaches, where the mapping among the different conceptual constructs of the involved metamodels can be 1:M, M:1, and M:M.

Finally, it is important to point that even though UML profile is the customization solution that better fits to the purpose of this thesis, this extension mechanism can be improved to obtain a sound support for model-driven interoperability. Works such as [60] show certain aspects that can be improved of UML and indicate how the UML profile specification has been evolving from previous UML versions.

Following, the OO-Method approach and the *i** framework are introduced. These approaches are used in the two application scenarios defined to verify and improve the proposed MDD interoperability approach.

2.2. The OO-Method MDD Approach

The OO-Method approach is an object-oriented MDD method successfully applied to the software industry [12]. OO-Method separates the business logic from the platform technology in order to allow the automatic generation of final applications by means of well-defined model transformations [61]. OO-Method performs this automatic code generation from a conceptual representation of the required software systems, which is defined by means of the OO-Method conceptual model.

The OO-Method conceptual model is centered on the specification of Management Information Systems (MIS) in a precise way and without ambiguity. This conceptual Model captures the static and dynamic properties of the system in a *Class Model*, a *Dynamic Model*, and a *Functional Model*. It also allows the specification of the user interfaces in an abstract way through the *Presentation Model*. From these four models that comprise the OO-Method

conceptual model, the class model is the most important, and the other models are defined (or derived) from this central model.

The automatic generation of the final software solution is performed starting from the OO-Method conceptual model by applying the OO-Method software production process that is presented in Figure 4. This production process consists in the automatic generation of an *Execution Model* [62] [63] from the OO-Method conceptual model by means of a set of model-to-model transformations.

The transformations performed to obtain the Execution model are related to transforming the constructs of the Conceptual Model into the corresponding software representations (i.e. the preconditions of services, the derivation formula for the derived attributes, the body of the services, the integrity constraints for classes, the valid state transitions, the filters formula for the presentation, etc.). Therefore, the execution model is configured according to the target implementation platform that is selected for the conceptual model compilation.

Next, by means a second model transformation (model-to-code) the generated execution model is automatically transformed into the final implementation model, which corresponds to the source code of the intended software system.

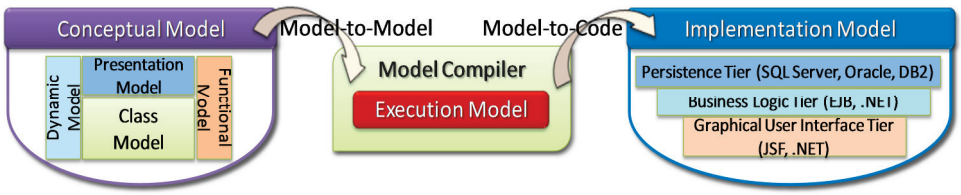

Figure 4. The OO-Method Software Production Process

The industrial application of OO-Method is performed by means of a suite of modeling tools and a model compiler developed by the company CARE Technologies. This MDD solution is called *OlivaNova The Programming Machine* [16, 64]. Figure 5 shows an interface provided by the Olivanova compilation tool, which is related to the selection of the different target implementation platforms supported by the model compiler, such as JSP, ASP, C#, EJB, SQL, ORACLE, DB2, etc.

The generated applications have a three-tier architecture: one tier for the client component, which contains the graphical user interface; one tier for the server component, which contains the business rules and the connections to the database; and one tier for the database component, which contains the persistence aspects of the applications.

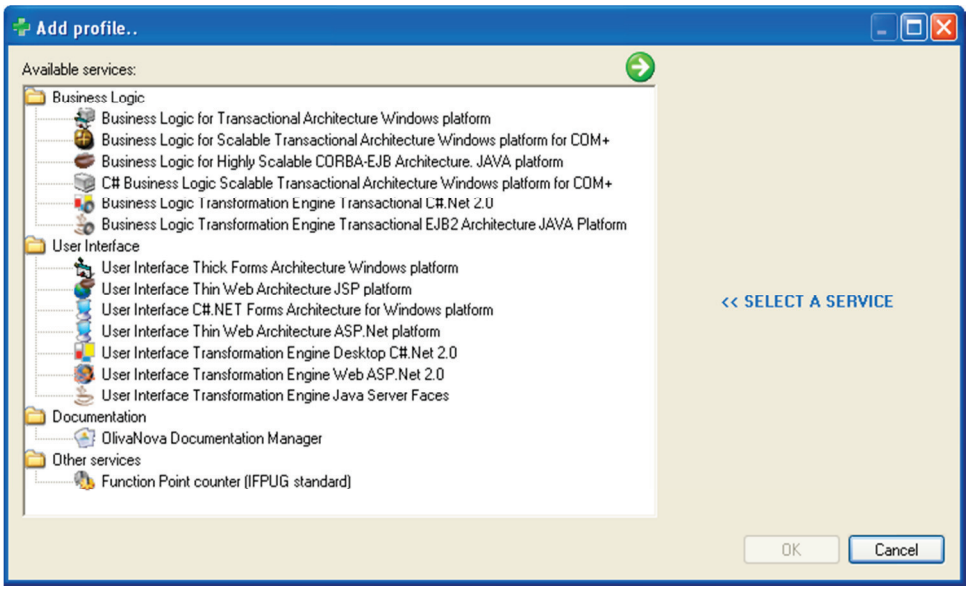

Figure 5. Compilation alternatives provided by the Olivanova technology

It is important to remark that the software generation performed by the OO-Method model compiler is obtained from a conceptual model that is independent of the implementation platform. Hence, a same OO-Method model can be used to automatically generate software products using different programming technologies and persistence platforms. With this, the relevance of defining appropriate conceptual constructs for the modeling languages that are related to MDD approaches is demonstrated. Following, the four models that comprise the OO-Method conceptual model are briefly explained.

2.2.1. The OO-Method Class Model

The class model of the OO-Method approach (that is also known as OO-Method object model) describes the structural part of the system. This model allows the specification of classes, attributes, derived attributes, events, transactions, operations, preconditions, integrity constraints, agents, and relationships between classes. The main concepts of this class model are well-known because most of they are the same as those used in the UML class model [65].

The main conceptual construct is the class. A class describes a set of objects that share the same specifications of characteristics, constraints, and semantics. A

class can have attributes, services, integrity constraints, and relationships with other classes of the model.

The attributes represent features of a class. The values related to attributes can also be derived from the values of other attributes or constants.

The services of a class are basic components that are associated with the specification of the behavior of a class. The services can be events, transactions or operations. The events are atomic services, indivisible, which can assign a value to an attribute. The transactions are a sequence of events or other transactions that have two ways to end the execution: either all involved services are correctly executed, or none of the services are executed. Finally, the operations are a sequence of events, transitions or other operations, which are executed sequentially independently of whether or not the involved services have been executed correctly. The services can have preconditions that limit their execution. The preconditions are conditions that must hold for the execution of a service.

The classes can also have integrity constraints, which are expressions of a semantic condition that must be preserved in every valid state of each object of the involved class.

The relationships between classes can be generalizations, binary associations, or agent relationships. The agent relationships are a particular construct of the OO-Method approach, which represents the visibility and execution properties that an object of the class model has over the attributes and services of other objects of the model.

2.2.2. The OO-Method Dynamic Model

The OO-Method dynamic model is comprised of two diagrams: the state transition diagram and the object interaction diagram. The state transition diagram defines the valid lives of the objects that belong to a class. This diagram has the following conceptual constructs: initial state, final state, intermediate states, and transitions. Most of the concepts of this diagram are the same as those used in the UML state transition diagram [65]. The initial state represents the state that objects are in immediately before they are created. The final state represents the state that objects are in immediately after they are destroyed. The intermediate states represent the set of possible stages of an object during its life.

The intermediate states have incoming and outgoing transitions, which represent a change of the state of an object. The transitions are activated by an agent that executes a service and can also have a condition to execute the service when it is required.

The object interaction diagram defines the interactions among the objects of the system. To do this, the triggers of the classes of the system and the global transactions or operations of the system are defined. The triggers are defined in a specific class. Each trigger is composed by a condition and a service to be executed. Each trigger service is executed at background; i.e., the result of a trigger execution (either success or failure) is not visible for the user.

The global transactions and operations are sequences of services, like the transactions and operations of the class model are. The global services can involve services of any class of the system. Usually, these services are defined when it is necessary to execute services of objects that are not related.

2.2.3. The OO-Method Functional Model

The functional model of the OO-Method approach allows the specification of the effects that the execution of an event has over the values of the attributes of the class that owns the event.

The functional model uses *valuations* to assign values to the corresponding attributes. The valuations can have preconditions. These preconditions and the effects of the valuation are specified by means of well-formed, first-order logic formulas.

The change that a valuation produces in the value of an attribute is classified into three different categories: *state*, *cardinal*, and *situation*. The state category implies that the change of the value of an attribute only depends on the effect specified in the valuation formula. The cardinal category increases, decreases or initializes the numeric-type attributes. The situation category implies that the valuation effect is applied only if the current value of the involved attribute is equal to a predefined value.

2.2.4. The OO-Method Presentation Model

In order to specify the interaction between the users of an application and the system, the OO-Method approach allows the specification of views in the object

model. A *view* corresponds to a set of interfaces, which are the communication point between agents (abstract representation of system users) and classes of the OO-Method class model. When the views of a system have been defined, the interaction model of each view must be specified.

The presentation model allows the specification of the graphical user interface of an application in an abstract way [66]. Thus, it is possible to completely specify the abstract graphical user interface of the intended applications. Then, the Model Compiler transforms the presentation model into the corresponding concrete user interfaces to characterize those parts of the final software product that represent the user interaction.

2.3. The *i** Framework

In general terms, the *Goal-Oriented Requirement Engineering* (GORE) approaches are oriented to obtain the 'why' of the intended systems through the analysis of organizational scenarios [15, 67]. Among several existing GORE approaches, the *i** framework [68] is one of the most widespread modeling and reasoning frameworks.

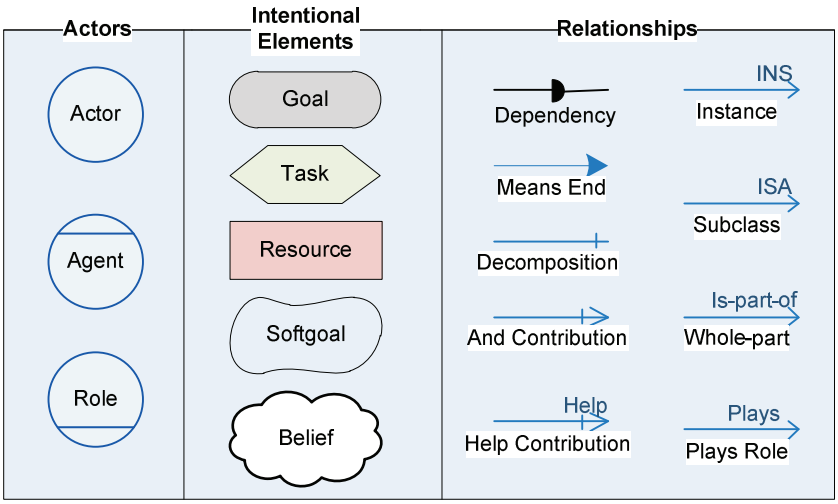

Figure 6. The *i** notation

The *i** framework has been originally defined by Eric Yu in [6]. It is focused on modeling and reasoning about organizational environments and their

information systems. It emphasizes the analysis of strategic relationships among organizational actors capturing the intentional requirements. The term *actor* is used to generically refer to any unit for which intentional dependencies can be ascribed. Actors are intentional, in a sense that they do not simply carry out activities and produce entities, but they also have desires and needs.

The *i** framework is comprised by two complementary models: the Strategic Dependency model, and the Strategic Rationale model. Figure 6 shows the notation related to the definition these *i** models.

2.3.1. The *i** Strategic Dependency Model

According to the definition presented in [69], the Strategic Dependency (SD) model is used to describe the dependency relationships among various actors in an organizational context. This model is focused on external relationships among actors, which are called dependencies. The SD model is defined by means of a set of nodes and connecting links, where nodes represent actors (*depender* and *dependee*) and each link indicates a dependency (*dependum*) between two actors. In the SD model, the internal goals, knowhow, and resources of an actor are not explicitly modeled. In this model, we distinguish among four types of dependency links that are based on the type of *dependum* element, which can be *goal*, *resource*, *task*, and *softgoal*.

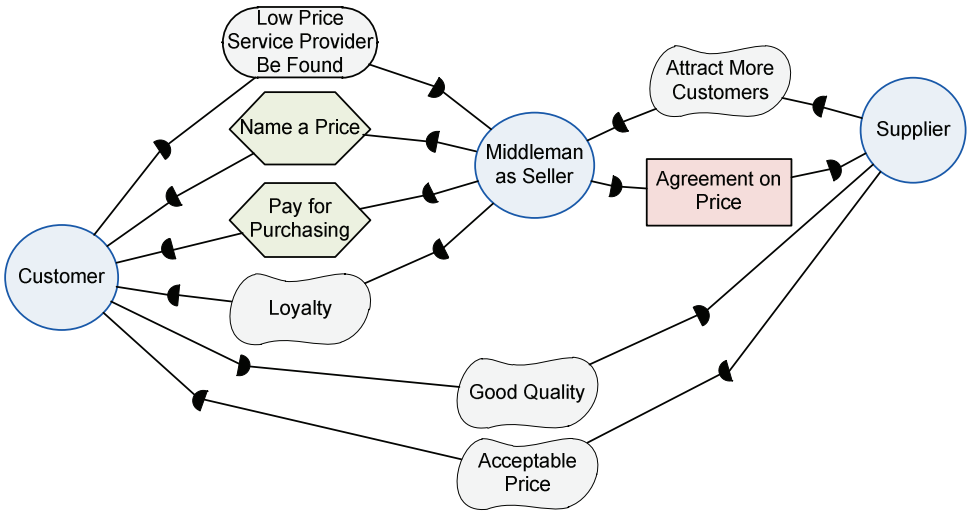

Figure 7. Strategic Dependency model of a buyer-driven e-commerce system

In the *i** context, a *goal* is a condition or state of concerns that an actor wants to achieve. A *resource* is a physical or informational entity that must be available for an actor. A *task* specifies a particular way of doing something, which can be decomposed in small sub-tasks. Finally, a *softgoal* is associated to non-functional requirements.

Figure 7 shows an example *i** SD model that has been defined by Eric Yu in [70]. This *i** model is related to a buyer-driven ecommerce system. In this system, the customer depends on a middleman to find a service provider who is willing to accept a price set by the customer. The customer submits a priced request to a middleman. The middleman forwards the request to suppliers. If a supplier decides to accept the request, it makes an agreement with the middleman. The middleman expects the customer to pay for the purchase in time.

2.3.2. The *i** Strategic Rationale Model

The Strategic Rationale (SR) model is used to describe stakeholder interests and concerns, and how they might be addressed by various configurations of systems and environments [69]. The SR model expands the description of a given actor and all rationales involved on its intentions, providing support for modeling the reasoning of each actor about its intentional relationships. In addition to the dependencies present in the SD model, three new types of relationships are incorporated in the SR model. These relationships are the following:

1. *Task-decomposition* links. These links indicate the elements that are necessary to perform a certain task.
2. *Means-end links*. These links indicate when a task is a mean to achieve a goal
3. *Contributions links*. These links indicate how a model element can contribute to achieve a *soft-goal*.

Summarizing, an SD model provides a general vision of *i** actors and their dependencies. An SR model is a detailed view of an SD model, which shows the internal elements related to the defined actors. These internal elements must be specified inside of the corresponding actor's boundary. Thus, the constructs of an SD model can be considered as a subset of the constructs of a SR model. Figure 8 shows the SR model for the example buyer-driven ecommerce system.

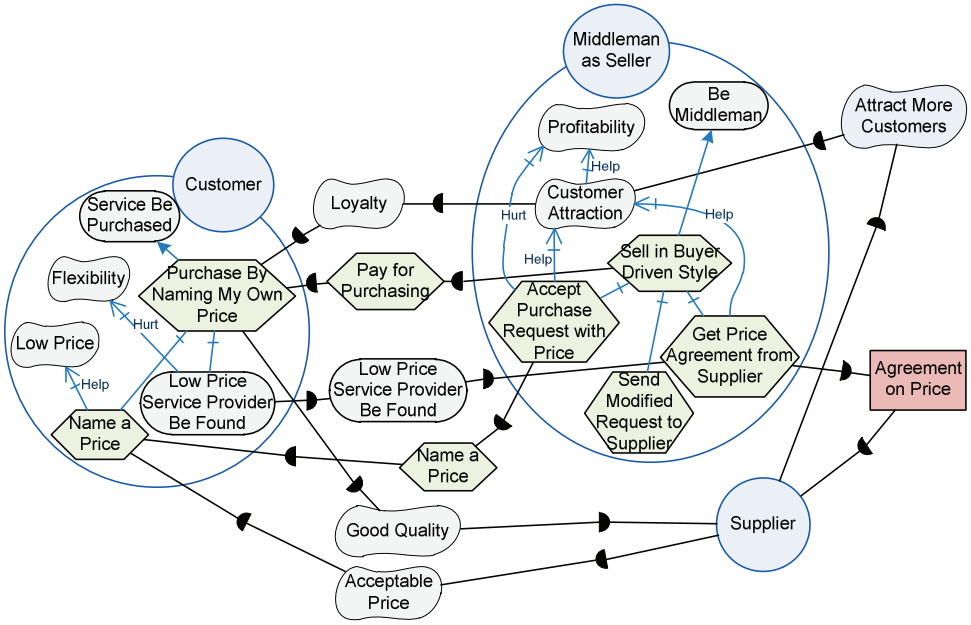

Figure 8. Strategic Rationale model of a buyer-driven e-commerce system

Chapter III.
Related Work

The emergence of several model-driven development (MDD) proposals requires the definition of proper interoperability mechanisms that facilitate the reuse of knowledge in the MDD community by taking advantage of already defined modeling languages, tools, and standards. However, there are no recent studies that cover the existing interoperability alternatives in the model-driven domain. This chapter confronts this situation through a systematic analysis of recent interoperability approaches that provide relevant features for MDD processes. The results of this analysis are presented in a general interoperability framework, which presents those aspects that are already covered by existing proposals as well as those pending subjects that, from our point of view, are future challenges to be faced in the model-driven interoperability domain.

The interoperability of multiple modeling approaches in different stages of a MDD process is a natural alternative for obtaining a proper software representation at different abstraction levels. This facilitates the integration of MDD approaches and already defined modeling languages, tools, and standards. The interoperability in MDD processes also facilitates the integration of different development groups in a common development project (e.g., software factories with different modeling approaches and tools [9]).

However, there is a lack of appropriate references that indicate and compare the current state of approaches that are related to supporting model-driven interoperability. Moreover, the proposals that define an interoperability framework for MDD processes are still in a theoretical space [71] and are not aligned with current standards, interoperability approaches, and technologies.

In this chapter, we deal with this issue by performing a systematic literature review of the existing proposals related to supporting interoperability in a model-driven domain. The results obtained from this systematic review are analyzed to indicate the interoperability aspects that have already been tackled by existing approaches as well as those aspects that are not completely solved and must be of concern in future research.

The rest of this chapter is organized as follows: Section 3.1 shows the planning of the systematic review. Section 3.2 presents the reviewed proposals by indicating their most relevant aspects and providing a summary of the different features analyzed. Section 0 presents our proposal for a common interoperability framework for MDD processes. Finally, Section 3.4 presents our conclusions and further work.

3.1. Planning the Systematic Review

The aim of this systematic review is to present an overview of the current state of Model-Driven proposals that provide relevant features for supporting the interoperability in MDD processes. According to Kitchenham [72], the following three phases are essential to perform an appropriate systematic review:

- Planning the Review: In this phase, the need of the review, the research questions, and the review protocol are defined.

- Conducting the Review: In this phase, the selection of primary studies, the quality assessment of the studies, the data extraction and monitoring, and the data synthesis are performed.
- Reporting the Review: In this phase, the results obtained from the review are reported.

The planning phase of the systematic review is presented in this section. The conducting phase is presented in Section 3.2. The reporting phase corresponds to the documentation of the review, which is presented throughout this chapter.

3.1.1. Research Question

In order to explore the evidence of model-based approaches with characteristics that support interoperability in MDD processes, we have formulated the following research question (RQ): What model-based approaches provide relevant interoperability features for model-driven development processes?

The features that we have considered to be relevant are the following:

- **Management of Heterogeneity (MH):** The proposal manages the structural differences that exist among modeling approaches that must interoperate.
- **Use of Standards (US)**: The proposal considers the use of existing standards related to modeling, metamodeling, or model transformation specifications.
- **Tool support (TS):** The proposal is presented at the theoretical level only, or by contrast, it has some kind of supporting technology.
- **Application Process (AP):** The proposal provides a well-defined process for its proper application.
- **Interoperability Verification (IV):** The proposal defines mechanisms to verify the correct interoperability among the modeling approaches involved.

The research question has been structured following the PICOC (Population, Intervention, Comparison, Outcome, Context) criteria [73]. The values for each criterion are the following:

Population: Model-based approaches.

Intervention: Interoperability approaches based on model interchange.

Comparison: The approaches are compared in terms of the features presented above.

Outcomes: Identification of a model-based interoperability framework.

Context: No context restriction.

3.1.2. Protocol

A review protocol specifies the method that is used to undertake a specific systematic review. A pre-defined protocol is necessary to reduce the possibility of researcher bias. In addition, the protocol must be reviewed and corrected several times by the systematic review team in order to obtain the best possible protocol. The contents of the final protocol used to perform the systematic review are explained in the following sub-sections.

Search Strategy for Primary Studies.

The search terms used in our systematic review were constructed using the following strategy: (1) Deriving major terms from the questions by identifying the population, intervention, and outcome; (2) Identifying alternative spellings and synonyms for major terms; (3) Using the boolean OR to incorporate alternative spellings and synonyms; and (4) Using the boolean AND to link the major terms. Thus, the construction of the search strings is based on the following terms:

Population: model, model-driven, model-based, conceptual model, domain model, modeling, diagram, schema, schemata.

Intervention: interoperability, interchange, integration, mapping, transformation, weaving.

Outcomes: framework, proposal, approach, schema, scenario, tool, process, solution.

The search process is organized in two stages (*first* and *second*): The *first stage* identifies candidate primary sources by using the search strings on the following electronic databases: IEEE Xplore, ACM Digital Library, Springer Link, Science Direct, Google Scholar. In order to assess the results of the search process, we compared the results with a small sample of primary studies that we already knew [74-78] in order to ensure that the search process was able to find the sample.

In the *second stage*, the references of the candidate primary sources identified in the *first phase* are revised to locate additional candidate primary sources. This process must be repeated until no further papers seem relevant.

Study Selection Criteria and Selection Procedures

The study selection criteria are used to determine which studies are included in, or excluded from, a systematic review.

The inclusion criteria were the following:

- The primary studies must have been published during the last five years (2007-2011). We consider that the proposals defined from the year 2007 on are based on current standards and technologies for modeling, metamodeling, and model transformations.
- The papers must present one or more features related to the research question.
- The papers must present results that are related to the interchange of information.

The exclusion criteria were the following:

- Approaches that present a pure model transformation or model integration.
- Older studies related to primary studies. We consider that the most recent publications provide updated versions of the contributions involved.

From the results obtained by the search strategy, the selection procedure is applied. First of all, duplicated studies had to be eliminated. Then, the title and abstract of the remaining studies were read by reviewers. If the nature of some studies was not clear, the reviewers also read the introduction and the conclusion of these studies. If the nature of the studies was still not clear, the reviewers had to read the entire study.

Data Extraction Strategy

The data extraction strategy defines how the information required from each primary study is obtained. This is a subjective process that, by nature, is error-prone. In order to minimize errors, we designed a template with 2 sections: general information, and specific information.

In the general information section, the following information is extracted from the studies and saved: title of the study, authors, conference or journal, volume and issue for journals, pages, and year of publication.

In the specific information section, the data extraction considers the features previously indicated for the research question: Management of heterogeneity (MH), Use of Standards (US), Tool support (TS), Application Process (AP), and Interoperability Verification (IV). Additionally, we extract the following two features:

- **Meta-Extensions (ME).** This feature is related to the definition of new information (extensions) in the involved modeling approaches to provide additional properties that are necessary by the interoperability approach.
- **Pivot Artifact (PA).** This feature is related to the use of an intermediate artifact (such as metamodel or ontology) to perform the interchange of information among modeling approaches.
- **Application Domain (AD).** This feature indicates the application domain that is related to the analyzed approach. It is important to remark that even though a proposal can be related to a specific domain, it has been selected because provide contributions that can be generalized in the MDD context.

Synthesis Extraction Strategy

The synthesis of the studies is presented in a table to facilitate the understanding of the information recovered. The table allows the readers identifying the similarities and differences among the studies selected in the systematic review.

3.2. Revision of Interoperability Approaches

In the selection of primary studies, 219 candidate primary studies were obtained after the revision of title and abstracts. Then, these candidates were analyzed by the reviewers applying the selection procedure, and the inclusion/exclusion criteria to finally obtain 33 primary studies. The review was finished on March 20th of 2011.

Following, the selected studies are briefly presented. We have grouped the revised approaches according to their main contribution (some approaches

provide more than one classification contribution). At the end of this section, a summary table (see Table 1) reports the results of the review.

3.2.1. Model Weaving

The model weaving approach is very popular in model transformation and model interchange contexts. It consists of the definition of a specific mapping model (called weaving model) among the metamodels of involved modeling approaches. The constructs of this weaving model are represented by means of a specific metamodel. Thus, weaving models indicate semantic equivalences through the definition of links among the metamodels' constructs. These links are considered as semantic connection points because they indicate those constructs (from the involved metamodels) that have an equivalent meaning (semantics) in the application domain. The definition of these links can be extended with additional information defined to manage structural differences among the linked constructs.

Fabro and Valduriez in [79] propose the use of weaving models between two metamodels to automatically infer model-to-model (M2M) transformations based on the ATL tool [80]. These transformations automate the translation between models defined with the involved metamodels.

The proposal presented in [78] by *Kappel et al.* is defined to support the interoperability among modeling tools. This proposal is based on a bridge metamodel (weaving metamodel) for the definition of semantic metamodel links (bridges). The defined bridges are used to transform the involved metamodels into equivalent petri-net representations. The petri-net representation is used to operationalize M2M transformations and to perform a formal verification of the structural differences (heterogeneities) among metamodels, which may produce interoperability conflicts.

The proposal presented by *Klar et al.* in [81] shows how the MDD interoperability can be used to support a complete development process. In particular, this proposal is centered on the integration of requirement modeling into the MDD process. However, it does not consider how to integrate specific aspects related to a particular MDD process into the requirement approach. These MDD aspects are necessary to obtain an appropriate requirement specification in the domain of the reference MDD approach.

The proposal presented in [82] by *Guerra et al.* consists of a pattern-based approach for defining bidirectional relations (a weaving model) among modeling

approaches. The main contributions of this proposal are the definition of specific inter-modeling patterns, which allow interoperability conflicts to be automatically identified. These patterns also facilitate the generation of model-to-model transformations, model matching, and traceability information.

3.2.2. Meta-Extensions

Seifert *et al.* in [83] indicate the advantages of using metamodels and model-to-model transformations to prevent the coupling among tools that must interoperate. This approach analyzes the pros and cons of proactive and retroactive tool integration alternatives. From this analysis, it suggests the use of a role-based metamodeling approach, which involves to extending the metamodels of tools with specific role information (defined in a role metamodel) to improve the tool interoperability.

The *BIZYCLE* framework applied by *Milanovic et al.* in [84] is defined to achieve applications and data integration by means of semantic annotations. These annotations are used to identify both semantic and structural conflicts that can be solved in a semi-automatic way.

The *Tran et al.* work [76] suggests the extension of the modeling approaches that must interoperate to specify the information related to a MDD process.

The proposal presented by *Agostinho et al.* [85] introduces an interoperability framework for business networks, which is based on UML for the definition of the involved metamodels. An interesting feature of this approach is the use of UML profiles to manage model heterogeneities and to obtain an appropriate model mapping. This work refers to those transformations that imply a structural change of the involved constructs as *model altering morphisms*.

3.2.3. Interoperability Verification

Radjenovic and Paige in [86] present an interoperability approach that is based on an initial identification of the issues that may prevent an appropriate model integration. This work considers both structural and behavioral interoperability conflicts. The detection of interoperability issues is performed by means of the transformation of the involved metamodels into a proprietary graph representation, which is called SMILE-X.

The proposal presented by *Polgár et al.* [87] indicates the need for interoperability in a common development process. In this approach, a reference

ontology is used to verify whether or not the involved modeling approaches are in conformance with the target development process.

3.2.4. Pivot Metamodel

The differences between a pivot metamodel and a weaving metamodel are related to their definition and use. A weaving metamodel is instantiated to represent links among constructs of the involved metamodels. From these weaving models, specific M2M transformation rules can be inferred according to a specific transformation approach, such as ATL [80], QVT [59], or ETL [88]. However, weaving models do not participate in the execution of the transformation rules. By contrast, a pivot metamodel can be a pre-defined representation of concepts (or constructs) for the interoperability domain or can be generated from the metamodels of the modeling approaches that must interoperate. Also, a pivot metamodel can be instantiated during the transformation process generating an intermediate pivot model.

The proposal presented in [9] by *Bruneliere et al.* defines a pivot metamodel to solve conflicts related to the heterogeneity among metamodels of modeling tools. This proposal is also based on current metamodeling standards and modeling tools.

The DUALLY approach presented by *Crnkovic et al.* [89] shows that the use of a pivot metamodel reduces the complexity of the necessary transformation rules. These rules can also be automatically inferred from the pivot metamodel definition.

The proposals presented by *Ziemann et al.* [90] and *Jankovic et al.* [91] are related to enterprise modeling. They use the POP* metamodel [92] as pivot metamodel. According to these proposals, the involved modeling approaches must be mapped to the POP* metamodel to determine common interrelation points. Similar approaches are presented by *Baumgart* [93] in the domain of embedded systems, and by *Mahé* [94] for visualization tools.

Berger in [95] considers the definition of a pivot metamodel that comprises all the conceptual constructs related to the modeling approaches that must interoperate. This pivot metamodel (defined as *generic metamodel* in the paper) is used as an interface among the metamodels of the involved modeling languages, which isolate the mappings from the metamodel heterogeneities. Later, by means of a set of pre-defined patterns, a model weaving among the involved metamodels is automatically generated to perform model-to-model transformations.

Vallecillo in [96] proposes the generation of a global model for the combination of different modeling approaches. The generation of this global metamodel is based on a viewpoint unification, which intends to comprise the benefits related to three metamodel integration techniques: metamodel extension, metamodel merge, and language embedding. The use of a common model obtained by the integration of the involved modeling approaches is also presented in the proposal of *Coutinho et al.* [97], which is related to organizational modeling.

In [98], *Moreno and Vallecillo* propose a web development interoperability framework, which is centered on a generic metamodel for web development methods. Thus, by means of QVT transformations, the modeling approaches related to the different development method must be mapped to the reference metamodel. Evidently, due to the use of QVT, the proposal requires that the involved modeling approaches be defined in MOF-compliant metamodels.

Diskin et al. [99] propose the generation of a pivot metamodel, which only indicates overlaps among different modeling approaches. This *overlap metamodel* reduces the complexity related to the definition of a big metamodel that covers all the modeling constructs of the involved modeling approaches. However, this proposal is still theoretical and is not supported by tools or standards.

In the work presented by *Biehl et al.* [100], the relevance of defining a bridge between technical spaces is clearly stated. This technical bridge is oriented to translating the metamodels of the involved modeling tools into equivalent representations that are defined using a common metamodeling language, i.e., this solves technical interoperability conflicts. Later, structural bridges are defined in the common interoperability space to perform M2M transformations among the metamodels generated by the technical bridges. A similar approach is defined by *Jouault and Guéguen* in [101]. In this approach, concrete modeling tools are translated into equivalent metamodeling representations, which are defined with a common metamodeling language. The resultant metamodel is called *virtual tool*. A similar view is presented by *Bambrilla et al.* [102], whose work proposes the translation of Domain-Specific Languages (DSL) to equivalent MOF representations.

In addition, in a previous work presented by *Lukácsy et al.* in [103], the outputs generated by different information sources are transformed into equivalent models (called *interface models*) to perform interoperability among web services. These interface models are expressed in a UML-like language. An improvement to this kind of service-oriented works is presented by *Tran et al.* in

[76]. In this paper, the authors propose a reverse-engineering mechanism to automatically infer model representation from services' views. The inference models are integrated in a view-based modeling framework to perform integration of services. These models are also used to generate code in other implementation platforms by following a MDD process.

The *ModelBus* approach presented by *Hein et al.* [104] is oriented to tool interoperability among nodes that participate in a common development scenario. The interoperability is performed by means of a repository of models and modeling services (such as model transformation and model verification services). This approach is based on the original idea of model bus presented in [105], which indicates two important aspects that must be considered to achieve the MDD interoperability: the *functional connectivity* (related to metamodel heterogeneity), and the *protocol connectivity* (related to technical heterogeneity).

3.2.5. Pivot Ontology

Höfferer in [106] presents an interesting analysis related to the use of ontologies and metamodels to achieve the model-driven interoperability. Even though this work is framed in the context of business-process modeling, the analysis and conclusions obtained can be generalized to any model-interoperability approach.

The *Sunindyo et al.* proposal [107] uses a common bus to perform the model-driven interoperability (such as in [104]). This proposal uses ontology mappings to identify common semantic links among different modeling approaches; the definition of these links is guided by a process for automatic discovery of the involved process models.

Roser and Bauer present in [74] an approach that uses an ontology specification (based on OntoMT) as an intermediate model for managing the heterogeneities and similarities among the metamodels of the involved modeling languages. It is also used to reuse the information of already defined M2M transformations, and to reduce the complexity related to changes in the versions of the involved metamodels. This approach distinguishes two kinds of model transformations: 1) mappings, which are horizontal model transformations defined at the same abstraction level; and 2) refinement transformations, which imply a change from a higher (less detailed) abstraction level to a lower (more detailed) abstraction level.

Other ontology-based approaches are the defined by *Berre et al.* [108], which is related to service interoperability; and the proposal presented in [109] by

Barnickel and Fluegge. This last work proposes the idea of semantic mediation at the domain level to improve the efficiency and the effectiveness of the ontology-based interoperability. The semantic mediation defines a pivot ontology for each involved domain, which groups a set of conceptual schemas. According to these authors, this approach provides a balanced interoperability solution, which is at a middle point between defining ontologies and mappings for each conceptual schema and the definition of a common pivot ontology.

Opdahl in [110] presents a modeling approach that is framed in the context of business processes. It facilitates language interoperability by applying the Unified Enterprise Modeling Language (UEML) [111]. This approach requires the translation of the involved DSMLs into the equivalent UEML representation.

Also in the context of business processes, the proposal presented by *Costa et al.* [112] provides a model-based platform for the enterprise interoperability. This proposal uses a reference ontology to identify semantic equivalences among the information (messages) that must interoperate. This information (defined in a XML format) is extended with annotations to manage heterogeneities in relation to the reference ontology, which are used to perform appropriate model-to-model transformations.

3.2.6. Threats to Validity

There are three main aspects in describing the validity of a study: construct validity, internal validity, and external validity.

The main threat to the construct validity was identified:

- **Selected databases are not representative to our study.** To minimize this threat, we selected the most representative electronic databases for the software engineering field (IEEE Xplore, ACM Digital Library, Springer Link, Science Direct, and Google Scholar).

The main threat to the internal validity was identified:

- **The systematic review was performed by only two people.** To mitigate this threat, the people that performed the systematic review had knowledge about conducting a review and interoperability of MDD approaches.

The main threat to the external validity was identified:

- **The results obtained cannot be replicated by other people.** To mitigate this threat, we detailed the review protocol; we systematically conducted the

review following this protocol; and we used the facilities that electronic databases provide to perform the search.

Table 1. Summary of the studies analyzed in the systematic review

Author	Year	MH	US	TS	AP	IV	ME	PA	AD
Agostinho [85]	2011	Y	Y	N	N	N	Y	N	Bussiness Networks
Barnickel [109]	2010	Y	N	N	N	N	N	O	Services
Baumgart [93]	2010	Y	N	N	N	N	N	M	Embeeded Systems
Berger [95]	2010	Y	N	Y	N	N	N	M	General
Berre [108]	2009	Y	Y	N	N	N	N	N	Services
Biehl [100]	2010	Y	Y	Y	N	N	N	M	Tool Interoperability
Brambilla [102]	2008	N	Y	Y	Y	N	N	N	Migration DSL to MOF
Brunelière [9]	2010	Y	Y	Y	Y	N	N	M	General (modeling tools)
Costa [112]	2007	Y	N	Y	N	N	Y	O	Bussiness Processes
Coutinho [97]	2009	Y	Y	Y	Y	N	N	M	Organizational Modeling
Crnkovic [89]	2009	N	Y	Y	Y	N	N	M	Component Models
Diskin [99]	2010	Y	N	N	Y	Y	N	M	General
Fabro [79]	2009	N	N	Y	N	N	N	N	General
Guerra [82]	2011	Y	N	Y	Y	Y	N	N	General
Hein [104]	2009	Y	Y	Y	Y	N	N	M	Tool Interoperability
Höfferer [106]	2007	Y	N	N	N	N	N	O	Bussiness Processes
Jankovic [91]	2007	N	N	N	Y	N	N	M	Enterprise Modeling
Joualt [101]	2009	Y	N	N	Y	N	N	M	Tool Interoperability
Kappel [78]	2011	Y	Y	Y	Y	Y	N	N	General
Klar [81]	2008	N	Y	Y	Y	N	N	N	RE
Lukácsy [103]	2007	Y	N	Y	N	N	N	M	Services
Mahé [94]	2010	Y	N	Y	N	N	N	M	Visualization Tools
Milanovic [84]	2009	Y	Y	Y	Y	Y	N	O	General
Moreno [98]	2008	Y	Y	Y	N	N	N	M	Web Development Tools
Opdahl [110]	2010	Y	N	Y	N	N	N	O	WebMl and UML
Polgar [87]	2009	Y	Y	Y	Y	Y	N	O	MDD
Radjenovic [86]	2010	Y	N	Y	Y	Y	N	N	General
Roser [74]	2007	Y	N	Y	N	N	N	O	General
Seifert [83]	2010	N	N	N	Y	N	Y	N	Tool Interoperability
Sunindyo [107]	2010	Y	N	Y	Y	N	N	O	Signal Engineering
Tran [76]	2008	Y	Y	Y	Y	N	Y	N	Services
Vallecillo [96]	2010	Y	N	N	N	N	Y	M	General
Ziemann [90]	2007	Y	N	Y	Y	N	N	M	Enterprise Modeling

Table 1 shows a summary of the primary studies analyzed by indicating the results obtained for the features that are involved in the data extraction strategy,

which correspond to the following: Management of heterogeneity (MH), Use of Standards (US), Tool support (TS), Application Process (AP), Interoperability Verification (IV), Meta-Extensions (ME), Pivot Artifact (PA), and Application Domain (AD). In the table, letters *Y* and *N* mean *Yes* and *No* respectively. In the *PA* column, the letter *O* means *Ontology* and the letter *M* means *Metamodel*.

3.3. A Common Interoperability Framework

This section analyzes the results of the review in terms of a common interoperability framework for MDD processes. There is no consensus in the terminology used by the proposals. For instance, the concept of pivot metamodel is called global model [96], generic metamodel [95], or overlap model [99]. Thus, a first challenge to confront is the alignment of concepts using a common terminology. With the proposed interoperability framework, we provide a first contribution in this direction. The general schema of this framework is presented in Figure 9.

An important aspect considered in this framework is the use of a pivot artifact. According to the review, 17 approaches (51.5% of the approaches) use a pivot metamodel and 7 approaches (21.2%) use a pivot ontology, which correspond to 72,7% of the total approaches analyzed. The use of a pivot artifact is oriented to managing structural heterogeneities through the definition of semantic links (weavings). We recommend the use of pivot metamodels for this purpose since the definition of weavings among metamodels and ontologies (also called lifting [113]) implies additional complexity. The work in [78] discusses in more detail the differences of using metamodel and ontology as pivots of interoperability processes. Furthermore, we consider the use of ontologies to be more appropriate for discovering semantic equivalences among modeling approaches, which can be used in the automatic generation of the pivot artifact and weavings. Nevertheless, this requires that the modeling approaches involved be supported by a standard ontological specification, which is another important challenge to be confronted in the MDD domain [78].

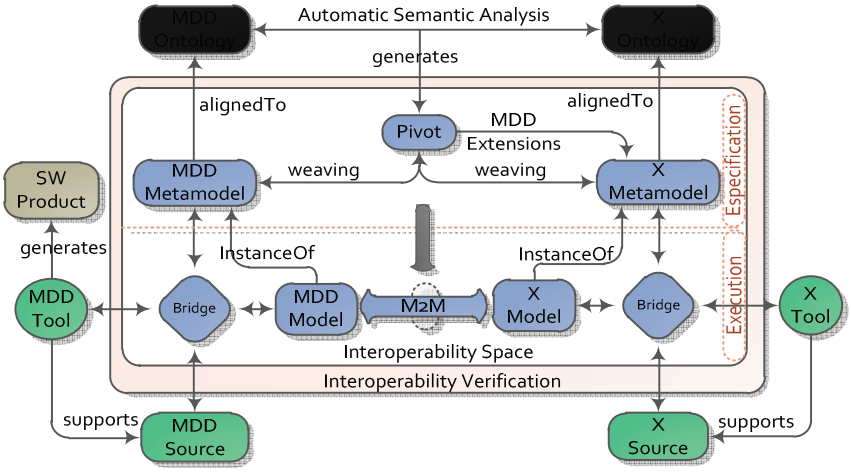

Figure 9. MDD-oriented Interoperability Framework

The use of modeling weavings is another important aspect to be considered in the framework, which separates the identification of semantic links from the final implementation of M2M transformations. The model weaving approach is put in practice by 21 (63.6%) of the analyzed approaches.

The use of a common interoperability space with a unique metamodeling language is an aspect that is considered by all the analyzed approaches. This may imply the redefinition of metamodels (or specific tools' interchange formats) from their original specification to the format used in the interoperability space. Hence, specific bridges, such as the ones presented in [100, 101], become necessary.

Also, we want to remark that only 4 proposals [76, 83-85] (12,1% of the approaches) indicate the need for defining modeling extensions to properly perform the model interoperability. We consider the use of metamodel extensions to be an important instrument in integrating specific MDD information in the involved modeling languages. Without these extensions, the necessary transformations cannot be totally automated, and human intervention is required. This is a time-consuming and error-prone task. These extensions also prevent the loss of necessary MDD information when the transformation of models is performed.

The verification of the interoperability is another aspect that has been partially tackled by the analyzed approaches. Only 6 approaches (18.2%) consider some kind of verification mechanism. These mechanisms are mainly focused on

an appropriate specification of mappings and model transformations (interoperability specification). However, none of the analyzed approaches consider the verification of the interoperability execution, i.e., the proper instantiation of the involved artifacts. In a real context, the defined models may contain errors or may not provide all the information necessary to perform the interoperability tasks, such as the execution of M2M transformations.

3.4. Conclusions

The results obtained from the systematic review clearly demonstrate that interoperability related to model-driven approaches is a hot research topic, which has received special attention over the last five years.

From the results obtained, redundancy in the works analyzed can be observed since different approaches tackle similar issues but are not related to each other. The systematic review presented here can be used as a reference point to contrast specific contributions with current approaches. Specifically for those novel interoperability approaches that focus on supporting MDD processes. In this context, we have defined a common interoperability framework for MDD, which is based on the results of this revision.

Despite the numerous interoperability approaches found, there still are important challenges that must be tackled, such as consensus in the terminology and concepts used, definition of mechanisms to facilitate or even automate the semantic mapping definition, appropriate mechanisms to guarantee correct interoperability definition and execution, or the definition and appropriate application of adequate support standards.

The conclusions obtained from the analysis of the different model-driven interoperability approaches are used for the definition of the MDD interoperability model and the interoperability process that are presented in the next chapter. Also, the performed systematic review has been relevant for the definition of the different interoperability artifacts that participate in our approach, which are presented throughout this thesis document.

Chapter IV.
Achieving MDD Interoperability

The definition of sound interoperability mechanisms to reuse knowledge and share ideas in the Model-Driven Development community is an important challenge to be faced during the next years. This kind of interoperability that is centered on model-driven techniques to combine different modeling approaches in a common development process is what we called MDD interoperability.

Several benefits can be obtained from MDD interoperability, for instance, the collaboration of MDD approaches related to different domains (such as software requirements, system design, business process, etc.), and the integration of different MDD tools and technologies (such as model compilers, or quality-assurance approaches).

This chapter presents a big picture about the aspects that we have considered to face this challenge, thus obtaining a specific process to support the intended MDD interoperability. The proposed process is based on a particular MDD interoperability model, which conceptualize the elements that are necessary to elaborate and automate an MDD interoperability solution.

In the current software context, the necessity of count with interoperability mechanisms at the different stages of a software development process is a growing trend. For instance, we can observe interoperability in web applications where different web services must be coordinated to perform a specific operation. Also, interoperability becomes necessary in development scenarios where geographically-distributed software factories are developing different components of a same software product [9].

According to the IEEE Standard Computer Dictionary [114], the interoperability is defined as *the ability of two or more systems or components to exchange information and to use the information that has been exchanged.*

In literature, it is possible to find several model-driven interoperability proposals, such as [115-118] (Chapter III shows a detailed analysis of the related work). The ideas defined in these proposals can be projected over the model-driven development contexts, where models (instead of programming code) are the key artifacts that must interoperate across the development process. This kind of model-driven interoperability that is focused on the interchange of modeling information (coming from different modeling approaches) throughout a common development process is what we called *MDD interoperability*.

Next, we present an MDD interoperability model and a set of challenges that we have faced to achieve the automatic MDD interoperability. For the application of the proposed MDD interoperability model a specific process has been defined. Thus, the stages and the interoperability artifacts involved in this MDD interoperability process are introduced at the end of this chapter.

The rest of this chapter is organized as follows: Section 4.1 presents our MDD interoperability model. Section 4.2 presents the challenges that we have faced to obtain the proposed MDD interoperability. 4.3 Introduces the process to complete and put into practice the proposed MDD interoperability model. Finally, Section 4.4 presents our conclusions.

4.1. An MDD Interoperability Model

Model-based proposals related to the context of information systems and tool interoperability state different levels [119] to achieve an appropriate interoperability framework, such as [120], [118], [116], and [121]. These

proposals have several common aspects by presenting similar interoperability levels. In particular, we have centered our attention on the LISI (Levels of Information Systems Interoperability) [118] and LCIM (Levels of Conceptual Interoperability Models) [116] because of their generic applicability, general acceptation, and maturity level. From the interoperability models presented in these two approaches, we have defined our specific MDD interoperability model.

The LISI approach proposes an interoperability model comprised of 5 levels (from 0 to 4). Level 0 (isolated interoperability) corresponds to a manual interoperability, where the interoperation tasks must be performed manually by the system users. Level 4 (enterprise interoperability) indicates that data and services are automatically interchanged by different applications in a transparent way for the system users.

The levels defined in the LISI model are transversally divided in four interoperability attributes called PAID, which correspond to Procedures, Applications, Infrastructure, and Data. Table 2 summarizes the LISI reference model.

Table 2. The LISI reference model

Interoperability	Computing Environment	Level	P	A	I	D
Enterprise	Universal	4	Enterprise Level	Interactive	Multi-Dimensional Topologies	Enterprise Model
Domain	Integrated	3	Domain Level	Groupware	World-wide Network	Domain Model
Functional	Distributed	2	Program Level	Desktop Automation	Local Networks	Program Model
Connected	Peer-to-Peer	1	Local/Site Level	Standard System Drivers	Simple Connection	Local
Isolated	Manual	0	Access Congtrol	N/A	Independent	Private

The LCIM approach is related to modeling and simulation, and, hence, it is closer to the model-driven development domain. Modeling aspects related to LCIM have a direct correspondence to the modeling tasks involved in MDD processes. Simulation corresponds to the execution of the modeled systems, therefore, it can be considered equivalent to the model compilation tasks that are involved in MDD processes.

The LCIM proposal states seven interoperability levels (from 0 to 6). Level 0 corresponds to the non-interoperability level (the same as the LISI proposal). Level 6 corresponds to the conceptual interoperability. In this level, interoperability among software systems is achieved by means of the definition of mappings among the conceptual models that describe the involved systems. In other words, conceptual interoperability is achieved through the meta-specification of the software systems. Figure 10 shows the levels defined for the LCIM model.

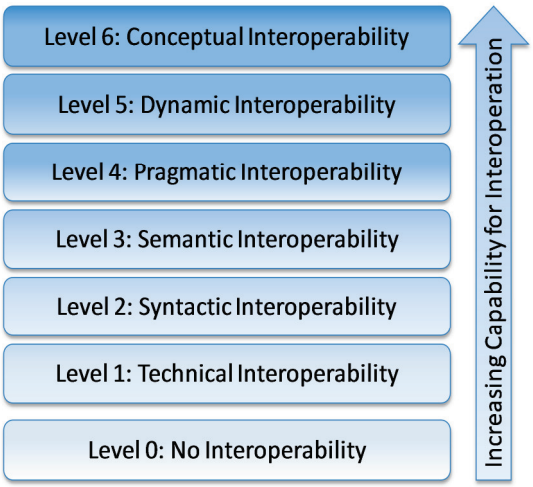

Figure 10. LCIM Model

If we project the LCIM ideas to the MDD context, we can state that to achieve conceptual interoperability in MDD process, it is necessary to define mappings among the involved MDD languages. To perform these mappings it is necessary to consider syntax, semantics, and technical aspects that are related to modeling languages definition, which corresponds to the levels 1, 2, and 3 of the LCIM approach.

The levels 4 and 5 (Pragmatic and Dynamic interoperability) of the LCIM approach are related to operation of information systems and management of systems' data in time. In a MDD context, these levels would be related to the evolution of the MDD models defined and their changes according to new system requirements. In this thesis, these interoperability levels are not considered since they are more related to model synchronization and model evolution, which are topics that are out of the scope of the developed work. However, these are two

interesting aspects that can be considered for future research in order to improve the MDD interoperability.

Thus, adopting the ideas proposed from the LISI and LCIM proposals, we have defined MDD interoperability as the interchange of modeling information among different modeling approaches that participate in a common model-driven development process. This MDD interoperability is comprised by semantic, syntactic, and technical interoperability (Levels of the LCIM proposal).

To automating MDD interoperability we have adopted the properties proposed by the LISI approach, which are the following:

- An appropriate interoperability *Procedure*, which indicates the elements that must be defined, and the steps that must be performed to interchange the modeling information.
- The *Applications* that manage the modeling information, which provide the features to automate interchange of models.
- The interoperability *Infrastructure*, which is related to the communication mechanisms among applications to assure the correct interchange of information, and to prevent the loss of modeling information when the interchange process is performed.
- The *Data* (modeling information) must be specified in a standard format, which can be interpreted by different modeling tools with independency of implementation platforms and development contexts.

In summary, the interoperability model defined (see Figure 11) states MDD interoperability in terms of Technical, Semantic, and Syntactic interoperability. Also, MDD interoperability can be automated by providing a concrete solution for Procedure, Application, Infrastructure, and Data properties.

In relation to syntactic interoperability, different modeling approaches have defined a particular syntax (abstract and concrete) to represent their modeling elements (conceptual constructs). This syntax is focused on supporting the semantics [48] of the modeling languages involved.

For the specification of the abstract syntax, it is possible to find standardized approaches, such as the Meta-object-Facility (MOF) [38]. The MOF approach provides suitable support for the generation of model-oriented technologies, such as model editors, and model transformation tools. This abstract syntax specification is performed by means of a metamodel definition, which represents

the different conceptual constructs (with their properties), the relationships that exist among the constructs, and a set of rules to manage the constructs' interaction.

From the metamodels that formalize the abstract syntax of modeling languages, the concrete syntax can be specified by using tools such as the Eclipse Graphical Modeling Framework (GMF) [50]. However, a standard for defining the concrete syntax related to a modeling language has not yet been defined.

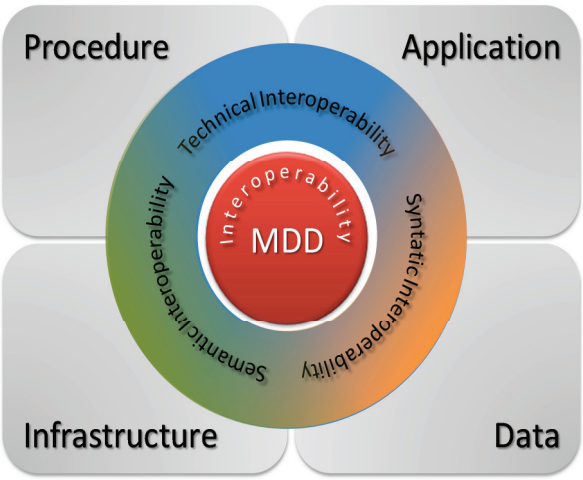

Figure 11. MDD Interoperability Model

The semantics related to modeling approaches is usually specified by means of textual representations, for instance, the UML specification [122] and the *i** framework [123]. In a MDD approach, we consider that the semantics is implicit in the mappings defined between the conceptual constructs and the corresponding software representations, which are used to perform the model-compilation process. However, there is a lack of an appropriate standard for the definition of the semantics related to modeling languages.

Thus, since only the abstract syntax of modeling languages is supported by standards that can be computationally interpreted, we propose the metamodels that formalize this abstract syntax as the starting point to support interoperability in MDD processes. From these metamodels, specific mappings can be defined (among the conceptual constructs of the involved modeling languages) to obtain semantic interoperability.

Technical interoperability can be achieved by using the interchange format defined for the open-source implementations of the metamodeling tools. For instance, the interchange mechanism implemented for the Eclipse UML2 tools is based on the XML specification [124]. This interchange mechanism is the XML Metadata Interchange (XMI) [40], which has been defined for the UML specification.

Thus, for automating MDD interoperability it is necessary: 1) to establish an appropriate *Procedure* to generate the interoperability artifacts; 2) to indicate or implement the *Applications* that are necessary to manipulate the models and perform the model interchange; 3) to state the *Infrastructure* that will be used to communicate the *Applications*; and 4) to define the format used for the modeling *Data* representation.

We have chosen open-source applications to support automatic MDD interoperability. In particular, we have considered the modeling tools developed in the context of the Eclipse Modeling Project [125], such as the Eclipse Modeling Framework (EMF) [57] and the Eclipse UML2 Project [126]. For the infrastructure, we have considered the XML implementation for the EMF tools, and the XMI specification related to the Eclipse UML2 models. The EMOF (EMF) and UML (Eclipse UML2) specifications provide the formats that are used to represent the modeling data. Thus, the applications, infrastructure, and data format are supported by current technologies, tools, and standards.

However, there is no a standard procedure that can be used to perform automatic MDD interoperability. Therefore we have defined a particular process, which indicates the steps and tasks that must be performed to automate the MDD interoperability. Figure 12 shows the proposed MDD interoperability model instantiated according to the defined process, and the considered standards and tools.

The process proposed for automating the MDD interoperability is center on the automatic interchange of modeling information, which can be performed by means of the integration of the modeling needs related to the target MDD approach into the involved modeling languages. To perform this integration, we consider that it is possible to represent the conceptual constructs that are required by specific MDD approaches (that are represented trough proprietary DSMLs) starting from conceptual constructs of already-existing modeling languages.

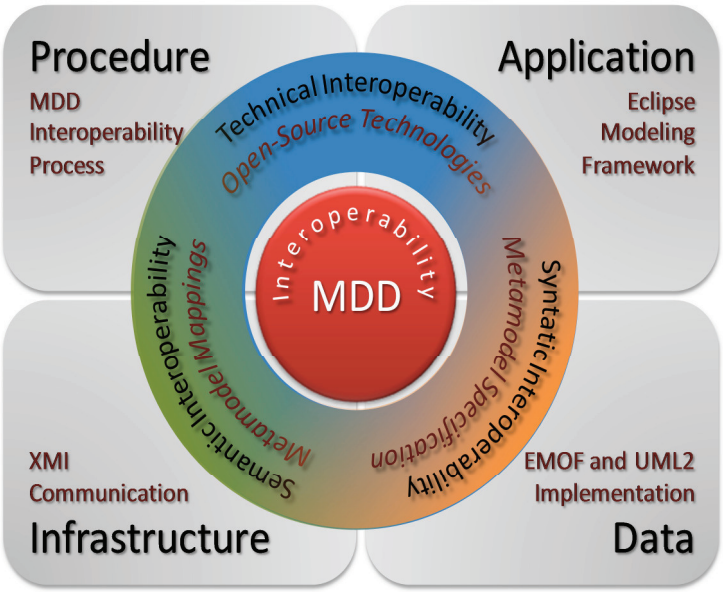

Figure 12. MDD Interoperability Model Instantiated

For an appropriate representation of the specific MDD constructs, the constructs of the target modeling languages are customized to fix differences or to add new properties in the context of the MDD approach. This is, to define modeling language extensions. For the implementation of the required modeling language extensions, we use the UML profile extension mechanism since it is a standardized extension mechanism that has been improved according to the UML experience, and it is based in the MOF metamodeling standard. Therefore, the fundamentals related to UML profile extensions can be generalized to any modeling language that uses a MOF metamodel to formalize its abstract syntax. In addition, UML profile is a lightweight extension mechanism that does not alter the target metamodel, and, hence, the defined extensions do not affect the compatibility with the technologies that are based on the original specification of the modeling language customized.

By analyzing the previous background and related work (see Chapter II and Chapter III), three challenges must be faced to support modeling language integration. Following, these challenges and the solutions proposed to solve them are detailed.

4.2. Challenges for Integration of Modeling Languages

The first of the three challenges that must be solved for modeling language integration is to indicate the modeling artifact that will be used as starting point for this integration. The second challenge is related to define an appropriate mechanism to indicate the semantic equivalences between the involved modeling languages. Finally, the third challenge is to automate the generation of the required metamodel extensions in order to reduce the potential errors and complexity that a manual modeling language customization involves.

4.2.1. First challenge: Establish the starting point

For solving this first challenge, we have considered the definition (or selection) of the metamodels that are related to the involved modeling languages as starting point. These metamodels are the artifacts were equivalences among the modeling languages can be identified and the required extensions can be defined.

Metamodels provide good support to formalize the abstract syntax of modeling languages, which is essential to perform an appropriate integration of modeling languages. Also, in the current MDD context, metamodels are widely used for development of technologies and modeling languages. Thus, it has sense to consider an element that is commonly used by MDD-oriented approaches as starting point of a MDD interoperability process.

The paper presented by Selic in [17] indicates a set of elements that must be considered for an appropriate metamodel specification. These elements are the following:

- The set of conceptual constructs related to the modeling language, which are defined as classes (metaclasses) of the metamodel.
- The set of relationships that exist among the different conceptual constructs.
- The set constraints that manage the interaction among the different conceptual constructs, which are necessary to define valid models (instances of the metamodel).
- The notation related to each conceptual construct when corresponds.
- The meaning of the conceptual constructs defined.

For the specification of the involved metamodels we propose the use of the MOF metamodeling standard [38]. The use of MOF facilitates the definition of

UML profiles for the implementation of modeling language extensions. Also, MOF is a suitable alternative for the specification of the required metamodels due to the following reasons:

- MOF is supported by a standardized interchange format (XMI [40]) .
- There exist different open-source metamodeling tools based on the MOF specification such as the Eclipse projects EMF [57] and UML2 [126].
- MOF is used by current model-to-model transformation technologies such as ATL [80] or QVT [59]
- There are many metamodel specifications based on MOF that can be used as reference modeling approaches.
- The use of MOF as common metamodeling language prevents the notation inconsistencies (at metamodel level) and facilitates the identification of equivalences between the different constructs.

However, there is an important lack of MOF for the appropriate metamodel specification, which is the impossibility of indicate the notation (concrete syntax) and meaning (semantics) of the defined constructs [48]. The MOF metamodels only specify the abstract syntax of the corresponding modeling languages. Therefore, the notation and meaning of the constructs must be documented in a separated way. This information is relevant for the correct metamodel specification and it is helpful to understand the defined metamodels. Also, the notation and semantics are relevant for the appropriate implementation of MDD tools, such as modeling tools and model compilers.

The MOF specification provides two alternatives for metamodel definition, i.e., two metamodeling languages. The first of these languages is the complete set of constructs of the MOF specification, which is called CMOF (Complete-MOF). The second alternative corresponds to a subset of the MOF constructs, which provide essential metamodeling facilities. This second metamodeling language is called EMOF (Essential-MOF).

The metamodeling capabilities that are provided by EMOF are closer to the extension capabilities provided by UML profiles. By contrast, CMOF provides a set of metamodeling facilities that cannot be represented by means of UML profile extensions, for instance, n-ary associations, or property redefinition. Therefore, we consider the use of EMOF to specify the metamodels of the involved modeling languages.

Once the corresponding EMOF metamodels are specified, or selected in the case of already existing EMOF metamodels, the equivalences between metamodels must be indicated. These equivalences are used to identify the necessary metamodel extensions. At this point, the second challenge that must be faced arises.

4.2.2. Second challenge: Identify Semantic Equivalences and Solve Integration Issues

This challenge involves the appropriate identification of semantic equivalences between the constructs related to a source and a target modeling language. In the context of our interoperability proposal, the source modeling language corresponds to the DSML that represent the constructs of the MDD approach involved, and the target modeling language is the pre-existing modeling language that will be customized with the specific MDD syntax. This identification of semantic equivalences can be performed by means of a mapping (or semantic links) between the constructs of the source metamodel and the target metamodel. Thus, this mapping guides the identification of the necessary extensions to integrate into the target metamodel the abstract syntax of the source metamodel. However, certain structural differences between the involved metamodels may prevent the appropriate mapping specification, and, hence, they prevent the correct identification of the required metamodel extensions. This situation is presented in works such as [17] and [19]. These works propose systematic approaches for the generation of UML profiles starting from metamodel mappings. However, due to structural differences that are present in the involved metamodels, the final UML profile generation cannot be completely automated. These structural differences also affect the completeness of the obtained UML profile, which cannot customize the target modeling language with all the modeling information required.

Therefore, to solve this challenge, we propose the definition of a pivot metamodel that allows the structural differences to be fixed, and an appropriate mapping specification to be obtained. This pivot metamodel is called *Integration Metamodel* [127] since it provides the necessary information to perform the appropriate integration of modeling languages.

4.2.3. Third challenge: Automatic Generation of Metamodel Extensions

Finally, the third challenge is related to how to automate the generation of the required metamodel extensions from the defined metamodels and the metamodel mappings. This is, to automate the generation of the required UML profile. The automatic generation of the required metamodel extensions prevents the potential inconsistencies between the syntax of the source and target metamodel that a manual specification may produce. In addition, the effort in the implementation of the UML profiles is considerably reduced due to the automatic generation since it is not necessary to known specific details related to the correct UML profile specification or deal with complexity of large metamodels. The benefits obtained from the automatic UML profile generation are very relevant since, according to Selic in [17], the lack of knowledge about the features of the UML profile specification has produced that many of the existing UML profiles definitions be invalid or of poor quality.

In general terms, the metamodel extensions that must be implemented in the UML profile can be automatically identified by comparing the source and target metamodels according to the semantic equivalences identified (defined in the metamodel mappings). Thus, the extensions are the additional modeling information that is necessary to fix the differences that exist between the target and source metamodel. For instance, if in the source metamodel there is a property that cannot be mapped to the target metamodel, then the UML profile extends the target metamodel with this non-mapped property.

Thus, a UML profile can be automatically generated by considering all the possible metamodel differences, and, for each one of these, to define specific rules that generate the necessary UML profile extensions.

Finally, the process for MDD interoperability is defined by considering the solutions proposed to solve the three challenges presented. This MDD interoperability process is detailed in the next section.

4.3. The MDD Interoperability Process

In this section, we introduce the process to achieve and automate interoperability in model-driven development, which is comprised by the following 4 steps: 1)

Definition of Modeling Language Metamodels; 2) Definition of Integration Metamodel; 3) Automatic UML Profile Generation; and 4) Generation of Model-Interchange Mechanisms.

The modeling language integration is the core of the MDD interoperability process proposed. It automates the generation of the necessary metamodel extensions and guides the specification of appropriate mappings, which are the main artifacts to perform the automatic model interchange. Thus, the first 3 steps of the interoperability process are related to perform the integration of the modeling languages involved. Figure 13 shows a general schema of the MDD interoperability process proposed, which is defined according to the BPMN notation [128].

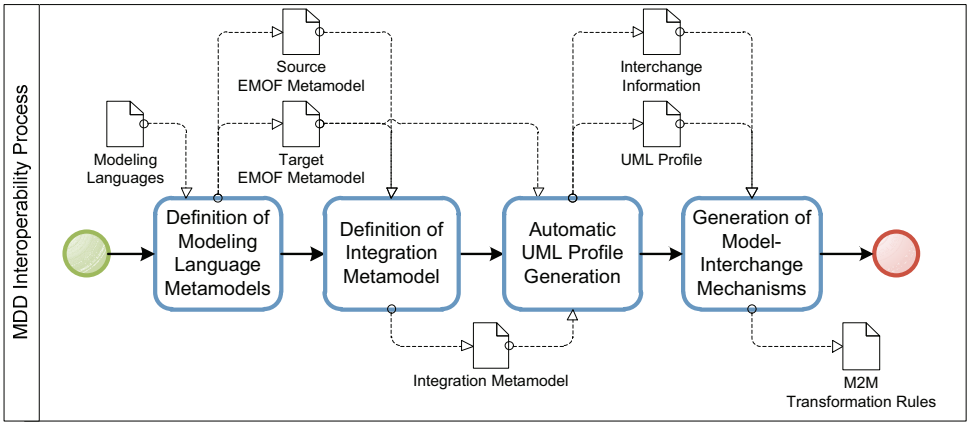

Figure 13. MDD Interoperability Process

In the definition of this interoperability process, different works have been considered. Some of these works are: definition of UML profiles using DSML metamodels [1, 19, 51, 54], correct use of metamodels in software engineering [129], UML profile implementations[1], interchange between UML profiles and DSMLs [130], and new UML profile features that are introduced in UML [131]. The 4 steps that comprise the interoperability process are detailed below:

- **Step 1: Definition of Modeling Language Metamodels.** The first step of the process corresponds to the starting point proposed as solution of the first

[1] OMG: Catalog of UML Profile Specifications, http://www.omg.org/technology/documents/profile_catalog.htm

integration challenge presented in the previous section, which is the specification or selection of the EMOF metamodels of the involved modeling languages.

- **Step 2: Definition of Integration Metamodel.** The second step is the definition of an Integration Metamodel to identify the equivalences between the metamodels involved and to fix the mapping issues that are produced by structural differences that may exist.
- **Step 3: Automatic UML Profile Generation.** This step considers the automatic generation of the UML profile that implements the metamodel extensions that are required to customize the abstract syntax of a target modeling language with the modeling information of the MDD approach involved.
- **Step 4: Generation of Model-Interchange Mechanisms.** This step considers the generation of the necessary model transformations rules to automatically obtain from the models that are defined with the customized modeling language appropriate inputs (models) for specific MDD tools, such as model compilers. The interchange mechanisms also transform MDD models into equivalent representation using the customized modeling language. This is a bidirectional interchange of models.

The different artifacts that are involved in the application of the proposed interoperability process are defined to facilitate the validation and verification in each step. Some of the validation and verification facilities that can be obtained are the following:

- It is possible to verify the abstract syntax related to the modeling languages thar must interoperate by means of the metamodels that are involved in the interoperability process. Also, the definition of metamodels by means of a standard metamodeling language (EMOF) facilitates the verification of the abstract syntax specified in relation to the supported semantics of the corresponding modeling languages.
- The construction of an Integration Metamodel facilitates the definition of specific rules for automatic UML profile generation. It allows the definition of verification mechanism that assure the correct application of these rules, and, hence, the correct generation of the resultant UML profile
- The interchange of models is based on specific model-to-model transformation rules, which are based on the generated metamodel extensions

and mappings. This allows the implementation of validation mechanisms to assure that the metamodel extensions and the defined models are defined according to the specification of the MDD approach involved.

4.4. Conclusions

In this chapter, we propose a specific MDD interoperability model, which is used as reference to identify the necessary tools, and artifacts to support the automatic MDD interoperability.

The different elements considered in the proposed MDD interoperability model have been instantiated by using the current model-based technologies. To complete the proposed interoperability model, we have defined a specific process, which indicates how the different elements of the proposed model can be coordinated to support the automatic interoperability in an MDD context.

Thus, the obtained MDD interoperability process is aligned with current modeling standards and MDD-oriented technologies, such as, modeling languages specification using metamodels, metamodels extensions mechanisms that are implemented as UML profiles, and interchange mechanisms that are implemented through models transformations. Also, time and errors related to manual specification of metamodel extensions and transformation rules are reduced by means of the automatic generation of the interoperability artifacts involved.

The structure proposed for the interoperability process is also a suitable reference for other metamodel extension mechanisms or proposals for model interchange. This structure is easily adaptable to different MDD-oriented technologies. For instance, the UML profile generation rules can be changed to implement the required extensions with a different extension mechanism. The work presented by Bruck et al. in [33] introduces different approaches for the definition of metamodel extensions and provides a comparative summary about the approaches that are presented.

The adaptation to potential changes that the involved modeling languages may suffer is also improved by the proposed interoperability process. Changes in the modeling languages directly impact in the defined metamodels. With the application of the proposed process, these changes are automatically propagated

to the interoperability artifacts (metamodels extensions, and mappings). This is very important, especially when the involved modeling languages are comprised by a big number of conceptual constructs, which are permanently changing. In this context, the manual identification of the impact that a change in the modeling languages has over the defined extensions and model transformation rules can be a titanic labor, which demands a lot of time and is very error prone.

Finally, the steps 2, 3, and 4 of the MDD interoperability process are based on original contributions that were created in the context of this thesis to tackle different interoperability challenges. These steps correspond to the Definition of an Integration Metamodel, the automatic generation of a UML Profile, and the Generation of model-interchange mechanisms. These three steps of the MDD interoperability process are detailed in the following chapters, which provide the information that is necessary to their proper application.

Chapter V.
The Integration Metamodel

In the context of MDD-oriented solutions, a modeling language with a precise semantics is a mandatory requirement. Nowadays, there are several MDD approaches that have defined proprietary Domain Specific Modeling Languages (DSML) to satisfy their modeling needs. However, there is an alternative modeling solution that considers the customization of existing modeling languages with the modeling needs of a specific MDD approach. Certain approaches provide alternatives to perform this customization by means of metamodel extension mechanisms. But, generally speaking, it is not possible to assure that the resultant extensions be properly defined according to the modeling standards, and that the customized modeling language includes all the expressiveness of the original DSML.

This chapter presents a solution to tackle the issues that may prevent a correct modeling language customization. This solution is based on a systematic approach to generate a specific DSML metamodel, called Integration Metamodel, which is used to automatically obtain a set of lightweight metamodel extensions that are implemented in a UML profile. As a result, a modeling language customized with the modeling needs of a specific MDD approach is obtained in accordance to the OMG modeling standards.

Generally speaking, the elaboration of UML profiles is a manual and intuitive task, which is performed without following a rigorous and well-defined process. This situation is motivated by the lack of a standard that specifies the correct way for the definition UML profiles and metamodel extensions [31]. For this reason, many of the existing UML profiles are invalid or of poor quality [17]. To avoid this situation, some works propose a more methodical solution that consists in the definition of a UML profile from the metamodel that describes the conceptual constructs required by MDD approaches. In other words, the UML profile is generated from a DSML metamodel [17]. This UML profile generation schema is based on the identification of the equivalences (correspondences) that exist between the source modeling language (which corresponds to a proprietary DSML) and the modeling language to be customized. The identification of equivalences is performed by means of a mapping between the different elements (classes, association, attributes, etc.) of the metamodel related to the source modeling language and the corresponding constructs of the metamodel of the target modeling language. Later, the identified equivalences are used to guide the correct definition of the required extension over the metamodel of the target modeling language through a UML profile implementation.

Additionally, from the experience acquired in the academic and industrial application of the OO-Method MDD approach [12], and the integration of OO-Method with other modeling approaches [132], we have identified three main requirements that must be fulfilled to obtain a proper process for the application of UML profiles in different MDD contexts:

- The UML profile definition must be automatic to reduce the complexity related to the correct UML profile specification, reduce the time that a manual definition requires, and to prevent the introduction of human errors. In particular, the two former risk factors (time and errors) are especially relevant in a MDD industrial context, where time costs money, and mistakes in implementation directly impact on customers' satisfaction.
- The UML profile must be easily adapted to the evolution of the MDD approaches since these are continuously changing in order to introduce new features that provide an appropriate support to the application domain and users' needs.
- The UML profile must integrate all the modeling expressiveness and precision that is required by the involved MDD approach in the target modeling language to provide a proper modeling framework for the involved MDD development process.

Certain proposals state that identification of equivalences between the source and target modeling language can be used to partially automate the UML profile generation [19, 54]. However, none of these approaches support the requirements mentioned above. In particular, these proposals cannot provide a totally automated solution for the generation of a complete UML profile since, in real MDD approaches, certain structural differences between the corresponding source and target metamodels may appear. These differences prevent the automated identification of all the metamodel extensions that must be implemented.

The automatic UML profile generation takes special relevance when the involved modeling languages have a large number of conceptual constructs. In this context, manual approach to determine the impact that the metamodel extensions have in the rest of related constructs and to assure the correct identification of all the required extensions is not suitable. The manual specification of a UML profile is a very error-prone and high time-consuming task [19].

This chapter presents the proposal defined in the context of this thesis to solve the structural differences that prevent the automatic identification of required metamodel extensions. This proposal is based on the definition of a specific metamodel that is named *Integration Metamodel*. The Integration Metamodel is defined from the metamodel of the source modeling language (the one related to the MDD approach) according to a set of specific rules by following a systematic approach. The structure of the Integration Metamodel allows a perfect integration with the metamodel of the target modeling language. The generation of this Integration Metamodel also implies the specification of the all the mapping information that is required to automatically generate a UML profile with all the modeling expressiveness and precision that the source modeling language has. This proposal has been defined by considering the different requirements presented before to provide a suitable approach that can be applied in different MDD contexts. Thus, it is possible to obtain an adequate input for an automated UML profile generation, which corresponds to the step 3 of the MDD interoperability process (see Chapter IV).

The rest of this chapter is organized as follows: Section 5.1 presents a brief background related to UML profiles definition. Section 5.2 explains why an Integration Metamodel is needed for the integration of modeling languages. Section 5.3 presents a systematic approach to obtain an Integration Metamodel. Section 5.4 presents a set of guidelines for the implementation of an Integration

Metamodel using current MDD technologies. Section 5.5 shows the benefits of the Integration Metamodel for the automatic profile generation. Finally, Section 5.6 presents our conclusions.

5.1. Background

In the literature related to defining UML profiles, two main working schemata can be observed: 1) the intuitive definition of the UML profile from scratch; and 2) the definition of a UML profile starting from a metamodel specification (a DSML metamodel). Although, different related works refer to the DSML metamodel used as source for the UML profile definition as *domain model*, in this chapter, the term *source metamodel* is used to avoid confusions between the concepts of model and metamodel. In addition, the metamodel of the modeling language to be extended is identified as *target metamodel*.

For the integration of modeling languages considered in this thesis, the second working schema has been selected, because it provides a methodological solution that provides more automation possibilities. In the brackground chapter (Chapter II) is presented a set of works related to this working schema. In general terms, all these works consider the UML metamodel as target metamodel, and not explore the generation of UML profiles for different modeling approaches. Also, all the analyzed works have some limitations to achieve a completely automated UML profile generation. These limitations come from the structural differences between the source metamodel and the target metamodel (UML metamodel in the related works). These differences prevent the definition of an adequate identification of equivalences between the two metamodels (pattern identification, mappings, etc.). This causes the resulting UML profile to provide different modeling expressiveness than the source DSML in the target modeling language and also it prevents the automatic UML profile generation. The next section briefly explains the relevance of having a correct mapping to automatically generate an adequate UML profile.

5.2. Improving the Integration of Modeling Languages

According to the background described above, the main problems in generating a UML profile from a source metamodel (related to a specific DSML) are the structural differences between the source metamodel and the metamodel of the modeling language to be extended (target metamodel). For this reason, the interoperability process proposed in this thesis requires the definition of a specific metamodel that has an appropriate structure and mapping to automatically generate the required metamodel extensions. This specific metamodel is called *Integration Metamodel*.

The Integration Metamodel has been specially formulated to automate the generation a UML Profile that integrates into a target modeling language all the modeling expressiveness of a source modeling language. In addition, the definition of an Integration Metamodel facilitates the generation of a correct UML profile, and reduces the effort related to manage the changes that can occur in the involved modeling languages.

The Integration Metamodel is a special DSML metamodel defined to automate the integration of a source metamodel (source DSML) into a target metamodel (target modeling language). The Integration Metamodel is defined from the source metamodel, and represents the same abstract syntax as the original metamodel. The main difference between the Integration Metamodel and the source metamodel is its structure since it is defined to obtain an appropriate mapping to the target metamodel. This mapping allows the automatic identification of the required metamodel extensions to be performed. The Integration Metamodel has the following features:

- It is defined according to the EMOF modeling capabilities that are defined in the MOF (Meta Object Facility) specification [38].
- It is mapped to the target metamodel by taking into account: Classes, Attributes, Associations, Enumerations, Enumeration Literals, and Data Types.
- All the classes from the Integration Metamodel are mapped to classes of the target metamodel. This assures the conceptual constructs of the source modeling language can be represented from the conceptual constructs of the target modeling language.
- The mapping information is specified by means of a mapping model that is based on a specific EMOF metamodel. Thus, all the information needed to

generate a UML profile is specified in XMI files that are defined according to the OMG Standards and can be processes by model-to-model transformation technologies.

During the Integration Metamodel definition, the original structure of the source metamodel may be redefined. This redefinition is performed without altering the abstract syntax represented in the source metamodel. This is illustrated by the example presented in Figure 14, which is oriented to integrate the abstract syntax presented in an example DSML metamodel (source metamodel) into the UML metamodel (target metamodel).

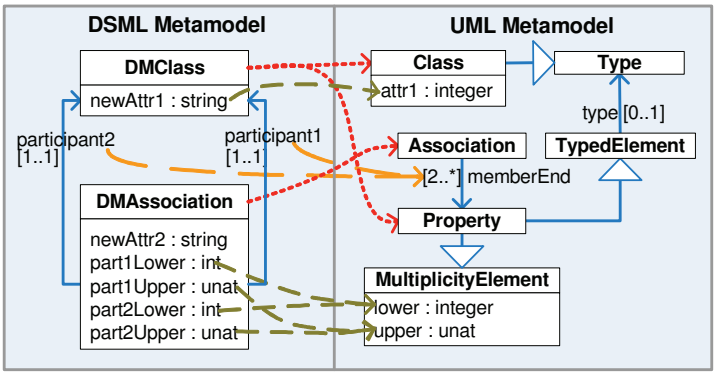

Figure 14. DSML metamodel example

Figure 14 shows a DSML metamodel that represents a binary association between classes. In this metamodel, the class (metaclass) *DMClass* represents classes of a model, and the class *DMAssociation* represents binary associations. The class *DMAssociation* has two associations that represent the two participant classes, *participant1* and *participant2*. This class also has four attributes to specify the lower and upper bound of each participant (association ends), these are: *part1Lower* and *part1Upper* for *participant1*, and *part2Lower* and *part2Upper* for *participant2*. In the two classes of the DSML metamodel, attributes for the generic representation of properties are defined. These attributes are *newAttr1* for the class *DMClass* and *newAttr2* for the class *DMAssociation*.

As Figure 14 shows, the DSML metamodel is mapped to the corresponding classes of the UML metamodel in order to define the equivalences (correspondences) that exist between the two metamodels. The mapping of the example (see Figure 14) indicates that the *DMClass* is mapped to the classes *Class* and *Property*. The mapping between *DMClass* and *Class* is clear because

both metaclasses represent classes of a model. The mapping between *DMClass* and *Property* is defined because, in UML, the participant classes of an association are represented by object-valued properties, where the types of these properties correspond to the related classes.

This mapping 1:2 (one construct of the source metamodel is mapped to two construct of the target metamodel) does not allow the automatic identification the extensions that are required to generate the corresponding UML profile. With this mapping, it is impossible to know if the attribute *newAttr1* (from *DMCLass*) must be considered to extend the class *Property* (by means of a tagged valued defined in the generated UML profile). Or by contrast, it is only an attribute related to the syntax of a class and must not be considered to extend the class *Property* for the correct representation of an association end.

The class *DMAssociation* presents a similar issue because this class is mapped to the UML class *Association*, but the properties related to cardinality are mapped to the UML class *MultiplicityElement*. In this case is impossible to determine if the properties related to cardinality must be considered, or not, to extend the class *Association*, which implies the generation of the corresponding tagged values.

The classes that participate in an association relationship are represented by two associations in the DSML metamodel (association *participant1* and *participant2)*, while, in the UML metamodel, this is represented by only one association (association *memberEnd*). With this mapping is impossible to know, in an automated way, if a new constraint is necessary to restrict the cardinality of the association *memberEnd*. Also, it is impossible to determine if a new association is necessary in the UML class Association to properly represent the participant classes according to the DSML syntax.

Furthermore, the type related to these associations that represent the participant classes is different in each metamodel. In the DSML metamodel the type corresponds to a class (metaclass *DMClass*), while in the UML metamodel the type corresponds to a property (metaclass *Property*). This difference cannot be directly managed by UML profile extensions because the tagged values cannot redefine UML properties to change the related type.

The proposal to solve these mapping issues is to restructure the DSML metamodel to align it with the UML metamodel, thereby obtaining the corresponding Integration Metamodel. Figure 15 shows the Integration Metamodel defined for the presented example.

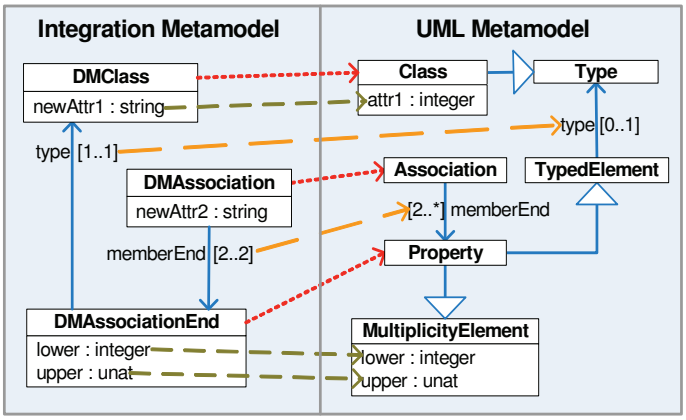

Figure 15. Integration Metamodel related to a binary association between classes

The elements of the Integration Metamodel that are mapped to UML elements are considered equivalent elements of the Integration Metamodel because these elements have correspondence (equivalence) in the mapped UML element. For instance in Figure 15, the class *DMAssociation* is equivalent to the UML class *Association*, and the attribute *DMAssociationEnd.lower* is equivalent to the UML attribute *MultiplicityElement.lower*.

In the Integration Metamodel presented in Figure 15, it can be observed that the conflicts related to the mapping of the class *DMClass* are solved by the definition of a new class called *DMAssociationEnd*, which is equivalent to the UML class *Property*. Thus, with the mapping obtained for the Integration Metamodel, it is clear that the attribute *newAttr1* is only involved in the representation of classes and it must not be considered to extend the class *Property*.

The new class *DMAssociationEnd* represents the participant classes of the association identified by means of the association type. The association *memberEnd* with cardinality [2..2] assures that an association only has two participants. Thus, the resultant Integration Metamodel represents the same abstract syntax as the DSML metamodel.

In order to facilitate the definition of the Integration Metamodel, a systematic approach to obtain the Integration Metamodel from an source metamodel has been defined [127]. This systematic approach is presented in the next section.

5.3. Systematic Definition of an Integration Metamodel

As the previous example has demonstrated, the identification of problematic mappings is important for the correct specifications of equivalences between the DSML and UML, and the final identification of the required UML metamodel extensions. However, performing a manual analysis for each defined mapping is not a suitable approach, especially in modeling languages with a large amount of conceptual constructs. In order to simplify this task, the identification can be automatically performed through a set of rules that establish specific constraints that must be fulfilled for a correct Integration Metamodel specification. These constraints guarantee a correct identification of the required UML extensions:

- **Rule 1:** All the classes from the Integration Metamodel must be mapped. This assures that the conceptual constructs of the source DSML can be represented from the conceptual constructs of the target modeling language.
- **Rule 2:** The mapping is defined between elements of the same type (classes with classes, attributes with attributes, and so on).
- **Rule 3:** An element from the Integration Metamodel is only mapped to one element of the target metamodel. These are X:1 mappings (with X greater than 0). For instance, many classes of the source DSML metamodel can be mapped to one class of the target metamodel. In this situation, the mapping rule is also accomplished because each class of the Integration Metamodel is only mapped to one class. However, two properties of a class of the Integration Metamodel cannot be mapped to the same property in the target metamodel. It is important to note that the many-to-many mappings that may exist between the original DSML metamodel and the target metamodel are transformed into X:1 mappings during the generation of the Integration Metamodel.
- **Rule 4:** If the properties (attributes and associations) of a class *A* from the Integration Metamodel are mapped to properties of a class *B* of the target metamodel, then the class *A* is mapped to the class *B* or a specialization of it. Figure 15 shows this situation can be observed in the mapping defined for the class *DMAssociationEnd* (from the Integration Metamodel), since the properties of this class are mapped to properties of the UML class *MultiplicityElement*, while the class *DMAssociationEnd* is mapped to the UML class *Property*, which is an specialization of *MultiplicityElement*. In

other words, a property is also a multiplicity element; therefore, the attributes of the class *MultiplicityElement* are also attributes of the class *Property*.

These four rules are the core of the systematic approach proposed for the Integration metamodel generation (see Figure 16). This systematic approach is composed of the following four steps:

- **Step 1:** Define the DSML metamodel related to the MDD approach, which corresponds to the source metamodel of the integration process. The definition of this metamodel must be elaborated according to the EMOF modeling capabilities [14]. EMOF provides a set of essential metamodeling constructs, and corresponds to a subset of the MOF specification. By using EMOF, the resultant DSML metamodel properties do not have features that are not supported by UML profiles. In the context of this thesis, the metamodel definition is performed within the first step of the MDD interoperability process. Thus, this initial step is not required when the systematic approach for the generation of the Integration Metamodel is applied in the MDD interoperability process (see Chapter IV).
- **Step 2:** Perform the mapping between the defined source metamodel and the metamodel of the target modeling language (for instance, the UML metamodel used in the example of Figure 14). The mapping must take into account: classes, properties (attributes and associations), enumerations, enumeration literals, and data types.
- **Step 3:** Verify that the OCL rules of the source metamodel do not produce conflicts with the OCL rules of the target metamodel. This validation is performed taking into account the OCL rules defined in the classes of the source metamodel and the equivalent classes of target metamodel (according to the mapping defined in Step 2). Since the OCL rules can be computationally interpreted, this validation can be automated by using works such as [133].
- **Step 4:** Fix mapping problems. Identify the elements whose mappings violate the constraints defined for a correct Integration Metamodel specification and modify its structure in order to fix them. The structure modification must consider the semantics that is supported by the corresponding elements of the source metamodel and the mapped elements in the target metamodel. This modification can imply the creation of new elements, or the modification of existing element. Then, for the new elements that are defined and the

elements that are modified, repeat the all steps of the systematic approach starting from the Step 2.

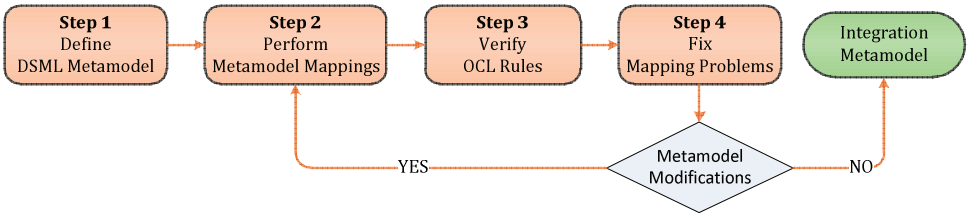

Figure 16. Systematic Approach for Integration Metamodel Definition

The systematic approach presented is iterative and finishes when all the mapping problems are solved. Figure 17 shows how the mapping problems that are present in the DSML metamodel presented in Figure 14 are solved to obtain the Integration Metamodel presented in Figure 15.

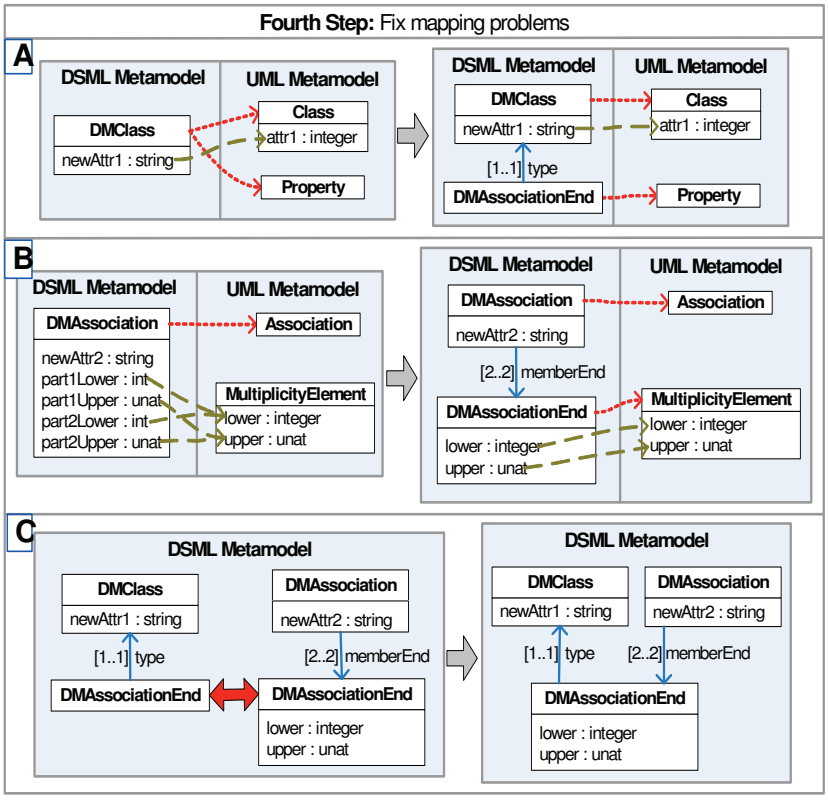

Figure 17. Fixing mapping problems for Integration Metamodel definition

In the Figure 17-A, it can be observed that the mapping defined for the class *DMClass* is indicating that exist double equivalence with the classes *Class* and *Property* of the UML metamodel. According to the Rule 3 of the set of rules defined for a correct Integration Metamodel specification, this 1:2 mapping is conflictive and must be fixed to perform an automatic UML profile generation. To perform this fixing, it is necessary to precisely indicate which part of the syntax of the class *DMClass* corresponds to the mapped classes of the UML metamodel. To do this, the original structure of the class *DMClass* is divided in two classes (one class per each mapped class of the target metamodel). Thus, a new class *DMAssociationEnd* is defined to indicate which part of the syntax of the class *DMCLass* is related to the UML class *Property*. The identification of this syntax is performed by considering the original semantics that is related to the construct *DMClass*, which, in this case, corresponds to the semantics of an association end. Furthermore, the association *type* between the generated class *DMAssociationEnd* and the class *DMClass* is defined to indicate the relationship that exists between these two classes. This relationship is also implicit in the original semantics of the construct *DMClass*.

Summarizing, in the original DSML Metamodel, the semantics supported by the class *DMClass* is the one related to class definition, but also, the semantics related to an association end definition. Thus, to fix the mapping problems, the abstract syntax related to the semantics of association end definition was separated from the class *DMClass* and explicitly specified by means of the new class *DMAssociationEnd* and the association *type*.

In figure Figure 17-B the situation is similar to that presented in figure Figure 17-A. Here, the class *DMAssociation* is mapped to the class *Association* of the UML metamodel, but the properties of the class *DMAssociation* are mapped to a different UML class than *Association*. These properties are mapped to the UML class *MultiplicityElement*, which is not a specialization of the class *Association*. According to the *Rule 4* related to a correct Integration Metamodel specification, this is an incorrect mapping and it must be fixed. In addition, according to the *Rule 3*, it is incorrect that two attributes of the class *DMAssociation* be mapped to one attribute of the class *MultiplicityElement*. Thus, to fix this mapping problems, the attributes related to association end *cardinality* are placed in a new class named *DMAssociationEnd,* which is mapped to the class *MultiplicityElement*. And the association *memberEnd* is defined to indicate the two association ends that are related to an association.

Figure 17-C shows that the new metaclasses defined to solve the mapping problems related to the classes *DMClass* and *DMAssociation*, are representing the same conceptual construct, the ends of an association. Therefore, the results obtained in Figure 17-A and Figure 17-B are combined to obtain an appropriate structure for the Integration Metamodel.

Finally, the second step of the systematic approach presented must be applied over the new class obtained (*DMAssociationEnd*) as well as the modified classes (*DMClass* and *DMAssciation*) in order to obtain the final Integration Metamodel (presented above in Figure 15).

It is important to remark that the systematic approach presented for the definition of the Integration Metamodel is iterative and finishes when all the mapping problems are solved.

Each time that the application of the fourth step of the systematic approach implies a change in the original DSML metamodel (such as in the example), then a mapping between the modified elements (the resultant Integration Metamodel) and the original elements (the DSML metamodel) is specified. This mapping information is used to validate that the resultant Integration Metamodel represents the same syntax as the original DSML metamodel. This validation consist into determine if, according to the mapping obtained, there could exist instances of the source metamodel that cannot have an equivalent representation with an instance of the Integration Metamodel; i.e., all modeling alternatives of the MDD approach must be supported by the Integration Metamodel.

Table 3. Mapping obtained from the Integration Metamodel generation

Integration Metamodel	DSML Metamodel
DMClass	DMClass
.newAttr1	.newAttr1
DMAssociation	DMAssociation
.newAttr2	.newAttr2
.memberEnd(1).lower	.part1Lower
.memberEnd(1).upper	.part1Upper
.memberEnd(2).lower	.part2Lower
.memberEnd(2).upper	.part2Upper

The generated mapping also allows the generation of mechanisms to transform an instance of the Integration Metamodel into an instance of the source metamodel (the DSML metamodel in the example). This mapping information is very valuable for the generation of the interchange mechanisms in the application of the interoperability process, which corresponds to the step four of this process.

The mapping information obtained for the example is presented in Table 3. This Table shows that the first and second instances of the association *memberEnd* represent the first and second participant classes of an association.

The obtained Integration Metamodel is the input required for the automatic UML profile generation (see Chapter 3), which corresponds to the third step of the MDD interoperability process proposed in this thesis.

5.4. Implementing the Integration Metamodel

The definition of the Integration Metamodel must be performed with a tool that supports the XMI standard [40]. Thus, it is possible to generate an XML [124] representation of the different MOF constructs that are involved in the metamodel definition. With this, a specific Integration Metamodel definition can be processes by different MDD technologies and tools based on the MOF specification, such as ATL or QVT model-to-model transformation technologies.

In general terms, most of the current UML tools support the XMI specification. However, it is important to mention that tool providers usually make modifications to the official XMI specification to introduce proprietary modeling features. For instance the XMI output that generates the UML tool provided by *Rational* tools differs from the XMI output generated by the *Poseidon UML* tool [132]. In this thesis, we have used the Eclipse UML2 Tool [2], which is an open-source project defined for the implementation of the UML standard, and, hence, it is compatible with the XMI specification for UML. We have chosen this UML tool because according to our experience this tool provides an XMI output that fulfills with the official XMI specification. Additionally, this tool has exportation facilities that allow the defined models to be exported as Ecore [2] models, which is the open-source implementation of the EMOF specification. Ecore is compatible with different open-source MDD technologies such as Eclipse ATL for the implementation of model-to-model transformations, or Eclipse EMF and Eclipse GMF for the implementation of

specific model editors. Furthermore, different MDD projects have been implemented by using these open-source technologies. For instance, all the tools developed in the context of the Eclipse Model Development Tools [134]. Thus, the use of open-source technologies that implement the current modeling standards facilitates the interchange of knowledge within the MDD community.

5.5. Benefits of the Integration Metamodel

The use of an Integration Metamodel for UML profile generation provides a set of benefits in relation to a direct transformation of the source metamodel or to the manual and intuitive UML profile definition. The most relevant benefits related to using an Integration Metamodel to generate UML profiles are following:

- *The definition of an Integration Metamodel is more intuitive than the direct definition of a UML Profile.* The Integration Metamodel definition is at the same abstraction level that the source metamodel. The Integration Metamodel is defined by using the same metamodeling language as the source metamodel (EMOF), which prevent the syntactical conflicts. Additionally, since the Integration Metamodel is defined independently of the UML profile implementation, it is not mandatory to understand the concepts involved in a UML profile definition; for instance, how to define stereotypes or the correct definition of extension relationships.
- *The Integration Metamodel helps to isolate the complexity related to a UML profile design decisions.* All these design decisions that are involved in a correct UML profile generation are defined in the transformation rules involved in the automated UML profile generation. Thus, the use of the Integration Metamodel helps to encapsulate the complexity related to the correct UML profile definition, but also, prevents the potential errors that a manual and intuitive UML profile specification should generate. Additionally, the effort to introduce changes in the UML profile (for instance, as a result of improvements performed to an MDD approach) is considerably reduced.
- *The Integration Metamodel provides a precise specification for the automatic UML profile generation.* The metamodel is defined according the MOF specification that can be processes by model-to-model technologies, and it

provides a structure that allows an appropriate mapping definition for the automatic UML profile generation. This is an important advantage of the Integration Metamodel vs. a direct mapping between the metamodels that are involved in the integration process. A direct mapping does not always provide enough information to automatically identify the required metamodel extensions [18, 135].

- *The Integration Metamodel allows the identification of required metamodel extensions independently of the target extension mechanism.* The structure of the Integration Metamodel and the related mapping to the target metamodel allow the automatic identification of the required extensions before the generation of the corresponding UML profile. This facilitates the verification of the final UML profile, which can be contrasted with the identified metamodel extensions. It also can be used to validate the correct definition of the involved transformation rules. In addition, it is possible to use the Integration Metamodel specification to automatically obtain different implementation of the required extensions, such as, for the generation of heavyweight extensions, or the automatic merge of the identified extensions directly in the target metamodel specification.

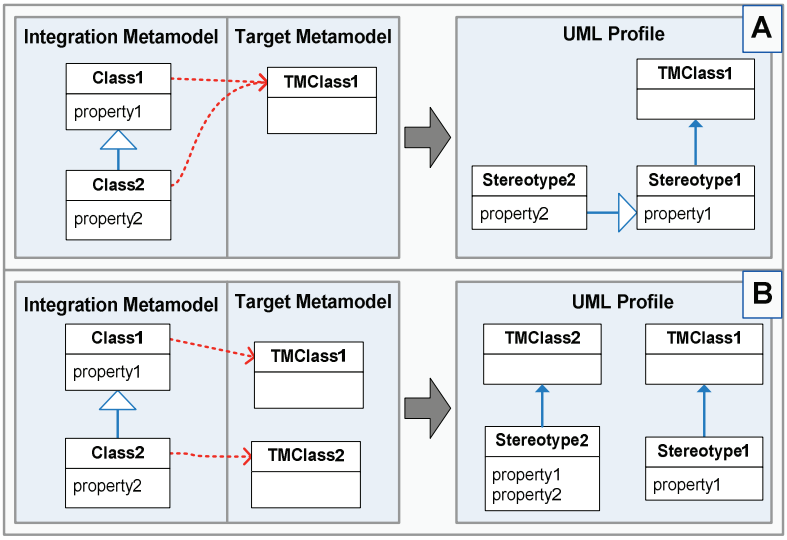

Figure 18. Example of a Integration Metamodel transformation

To exemplify the benefits mentioned above, a briefly transformation exercise is performed over the hypothetical modeling situation that is presented in Figure

18. This figure shows a subset of a generic Integration Metamodel and the correspondent UML profile that is generated in the transformation of it. In the generated UML profile, the stereotypes *Stereotype1* and *Stereotype2* represent the classes *Class1* and *Class2*, respectively.

The transformation presented in Figure 18-A has been carried out according to the following two transformation rules[2]:

"One Stereotype for each class of the Integration Metamodel. The stereotype extends the target class according to the mapping defined".

"Define one generalization between two stereotypes that represent equivalent classes that are associated with a generalization in the Integration metamodel and that are referencing the same UML class. The extension relationship related to the child stereotype is not defined since it is implicit in the generalization relationship".

In Figure 18-B, the mapping defined for the Integration Metamodel presented in Figure 18-A has been changed. This change generates a different profile according to the following transformation rule:

"If there is a new generalization relationship between two classes from the Integration Metamodel that are equivalent to different classes of the UML2 Superstructure, the generalization relationship is not represented in the profile, the extensions of each stereotype to the correspondent UML class are defined, and the inherited non-equivalent properties are duplicated"

In Figure 18-B it is possible to observe that a single change is introduced in the Integration Metamodel: the mapping of the class *Class2* is changed to a different UML class (Fig. 5-B). However, this change produces the following four changes in the resultant UML profile:

1. The generalization relationship is dropped
2. The UML class *UMLClass2* is imported
3. The extension for *Stereotype2* is defined
4. The property *property1* is added to the *Stereotype2*

[2] The complete set of transformation rules for the UML profile generation is presented in Chapter VI

This basic example shows that it is more intuitive and simple to change a mapping in the Integration Metamodel than to directly perform the required changes in the target metamodel. This is because it is not necessary to know implementation aspects of the UML profiles, the complexity related to the generation of the new UML profile according to the changes performed is encapsulated in the transformation rules, Furthermore, one simple change in the Integration Metamodel may involve many changes in the UML profile; for instance, in the presented example, one change in the Integration Metamodel has produced four changes in the corresponding UML profile. Also, the effort to obtain a new profile definition once the change has been performed is considerably reduced because the automatic execution of the transformation rules.

It is important to note that in an Integration Metamodel designed for real MDD solution, the impact of changes in the MDD approach can be even much greater than the presented example. This justifies the additional effort necessary to define the Integration Metamodel in order to automate the UML profile generation and reduce the time and potential errors that a manual definition involves.

5.6. Conclusions

In this chapter, an Integration Metamodel that improves the automatic generation of a UML profile has been introduced. The main purpose of this metamodel is the integration of the abstract syntax that is related to a source modeling language into a target modeling languages by means of the automatic generation of the necessary metamodel extensions. These extensions are implemented through a UML profile definition.

The Integration Metamodel reduces the effort of implementing the necessary UML profile and facilitates the adaptation of generated UML profiles to the evolution of the MDD approach involved. In addition, the proposed systematic approach facilitates the generation of the Integration Metamodel from the DSML metamodel of the MDD approach.

Apart from to the benefits mentioned, the Integration Metamodel can also be used as a mechanism to share knowledge between different MDD approaches because the structure and the mappings of the Integration Metamodel are aligned with the target metamodel, which can be a standardized modeling language.

Therefore, a better understanding of the semantics and design decisions involved in different proprietary MDD approaches can be achieved taking the target modeling language as reference. It is important to note that the concepts and ideas presented in this chapter can also be applied to improve those MDD approaches that are already using UML and profiles as modeling mechanism.

Chapter VI.
Automatic UML Profile Generation

According to our experience, the use of UML profiles is a recommended strategy to customize modeling languages in order to integrate specific modeling aspects. This integration allows existing modeling technologies to interoperate each other in the application of specific MDD approaches. However, in the literature related to UML profile construction; it is not possible to find a standardized UML profile generation process.

Therefore, this chapter presents a process defined to integrate a particular DSML into a target modeling language through the automatic generation of a UML profile. This process facilitates the correct use of existing modeling languages in a specific MDD context and provides a solution to take advantage of the benefits of standardized modeling languages and proprietary DSMLs.

An appropriate modeling language is one of the most important concerns for Model-Driven Development (MDD) approaches [1]. To obtain modeling languages that are adequate, different MDD approaches have defined their own Domain-Specific Modeling Languages (DSML) in order to represent their particular modeling needs. Two of the benefits that the use of DSMLs provide to MDD approaches are: (1) a correct and precise representation of the conceptual constructs related to the application domain, and (2) simplification of the implementation of tools oriented to improving the modeling tasks, development, and maintenance of generated software solutions.

Nowadays, different modeling languages are arising as standard (or de-facto standard) proposals for general-purpose modeling (such as UML) or for the application in specific modeling domains (such as *i** for requirement modeling). This has motivated that many MDD approaches integrate their modeling needs into these pre-defined modeling languages in order to use them as DSML. To perform this integration, the use of the metamodel extension mechanisms is the most suitable strategy by considering the current MDD standards and technologies. With the use of existing modeling proposals, especially those standardized proposals, the MDD approaches could achieve a larger market (greater number of potential users), take advantage of the existing modeling and MDD technologies, and reduce the learning curve [17, 33, 54]. In addition, the standard modeling languages can be used as a mechanism to interchange ideas and theories among different research communities.

We focus on the use of UML profiles as metamodel extension mechanism since UML profile is a lightweight extension mechanisms that do not alters the original metamodel specification of the extended modeling language. In addition, UML profile has a standard specification [122], it has a standard interchange format [40] (XMI specification for UML profiles [40]), and it has tools supporting [126]. Currently, there are many definitions of UML profiles that are associated to MDD approaches [43]. Generally speaking, these profile definitions are manually elaborated in a straightforward way and without a standardized process because a standard that specifies how the UML extensions must be defined does not currently exist [31]. For this reason, many of the existing UML profiles are invalid or of poor quality [17]. In addition, the manual definition of a UML profile is an error-prone and time-consuming task [19]. These two risk factors (time and error) must be avoided, especially in MDD approaches that are applied to industrial context, where time costs money and mistakes in implementation directly impact on customer satisfaction.

To avoid the risks described above, some works related to UML profile elaboration have defined proposals to achieve a semi-automated profile generation [19, 54]. For the generation of the UML profile, these proposals use as input the metamodel that describes the *conceptual constructs* related to the DSML of an MDD approach (the *DSML Metamodel*). However, none of these proposals provide a sound solution for the automatic generation of a complete UML profile. This is because, in real MDD solutions, structural differences between the DSML metamodel and the UML Metamodel, which prevent the automated identification of the extensions that must be performed in UML, may be found.

This chapter introduces a solution for a completely automated UML profile generation using as input the Integration Metamodel [8] related to a MDD approach. The details about Integration metamodel rationale and formulation are presented in Chapter V. The automatic UML profile generation presented in this chapter is part of the interoperability process that has been presented in Chapter IV.

Thus, this chapter shows how the required metamodel extensions can be automatically identified and details the transformation rules to obtain the UML Profile that implements these extensions. Additionally, in order to exemplify how the automatic UML generation can be used to integrate existing modeling approaches and proprietary DSMLs in a unique MDD solution, a brief example that is based on the integration of the industrial implementation of the *OO-Method approach* [12, 16] and UML is presented.

The rest of the chapter is organized as follows: Section 6.1 shows the background related to UML profile generation. Section 6.2 introduces the proposed process. Section 6.3 details the automatic UML profile generation. Finally, Section 6.5 presents our conclusions and further work.

6.1. Background

The UML profile is an extension mechanism that is specified in the UML Infrastructure [131], which is oriented to adapt existing MOF metamodels to specific platforms, domains, business objects, or software process modeling. In the context of this thesis, the UML profiles are used to integrate the modeling needs of MDD approaches in target modeling languages.

In the literature related to the definition of UML profiles for MDD approaches, two main working schemas can be observed: 1) the definition of the UML profile from scratch; and 2) the definition of the UML profile starting from a *DSML Metamodel* [17], which is the metamodel that describes the conceptual constructs required by a MDD approach. For the automatic UML profile generation presented in this chapter, the second working schema has been selected since it provides a methodological solution that has more automation possibilities. In Chapter II is presented a detailed analysis of the current approaches for generation of UML profiles from DSML metamodels.

In general terms, the analyzed works are only based on the UML metamodel as target metamodel, and they are centered on representing those modeling elements that are required by the MDD approach and that do not exist in UML using the generated UML Profile. However, this focus is not enough to generate a correct UML profile because there are other elements that must be considered for a correct UML profile definition. These other elements are: 1) the representation of the differences that exist between elements of the DSML, and corresponding elements that already exist in the target metamodel, and 2) the definition of rules oriented to validate the correct use of the UML profile in order to produce correct conceptual models.

Even when these additional considerations are omitted, none of the existing approaches for UML profile generation provide a sound transformation process to automatically obtain a complete UML profile. The main limitation of these approaches comes from the structural differences between the source DSML metamodel and the target metamodel. If these structural differences are solved, then the UML Profile generation can be automated. The Integration Metamodel presented in Chapter V presents a solution to solve these structural problems, which consists of the transformation of the DSML metamodel into a new metamodel called Integration Metamodel. This Integration Metamodel provides an adequate input to automate the integration of the abstract syntax that is represented in a source DSML metamodel into a target Metamodel. Thus the automatic UML profile generation that is presented in this chapter is based on this solution.

The UML profile generation is a process that is comprised by two steps: 1) the comparison of the metamodels to obtain the required metamodel extensions; and 2) The transformation of the Integration Metamodel into the corresponding UML profile. These steps are graphically represented in Figure 19 and are detailed in the following sections.

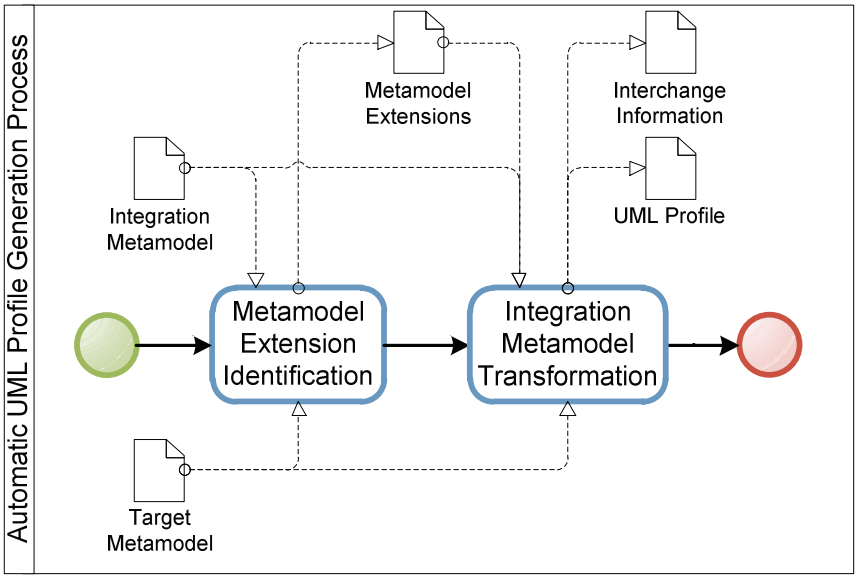

Figure 19. Automatic UML Profile Generation Process

6.2. Identification of Metamodel Extensions

The identification of the metamodel extensions that must be defined in the target metamodel is performed through a comparison between the Integration Metamodel and the target metamodel. To perform this comparison, the mapping information defined in the Integration Metamodel is used.

It is important to remark that in spite of the metamodel comparison is oriented to identify the metamodel extension that must be implemented in the final UML profile, this step is performed with independence of the aspects involved in the UML profile definition.

The comparison between the Integration Metamodel and the target Metamodel considers the following elements:

- The identification of *new elements*, which are the elements from the Integration Metamodel that are not equivalent to elements of the target metamodel. These elements can be attributes, associations, enumerations, literal values, and data types.

- The identification of differences in type or cardinality of *equivalent properties*, which correspond to attributes or associations that have equivalence in the target metamodel.

The identification of new elements is very simple to be performed; these are the elements of the Integration Metamodel that are not mapped to the target metamodel. The detection of differences in cardinality of equivalent properties is also very simple to detect. In this case the cardinalities established for equivalent properties are compared to the cardinalities of the mapped elements of the target metamodel. However, the identification of differences in type of equivalent properties requires additional explanation to properly understand how it must be performed.

In an EMOF metamodel, the type of a property can be stated by the following elements:

1. By a class of the metamodel, when the property corresponds to an association end.
2. By an *enumeration* or *data type*, when the property corresponds to an attribute.

For an association end, a type difference exists when the class related to the type of the source association end differs from the class related to the type of the target association end. Figure 20 shows an example of an equivalent association without difference type. In this figure, the type of the source association end (association1) is given by the class *Class1*. According to the defined mapping, the type of *association1* is the same than *TMassociation1* since *Class1*, which is the type of *association1*, is equivalent to *TMClass1*, which is the type of *TMassociation1*.

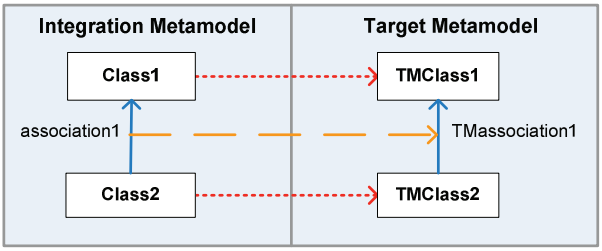

Figure 20. Example of an equivalent association without type difference

However, if Class1 is mapped to a different target class, then a difference type between the source and target association ends is present. Figure 21 exemplifies this situation.

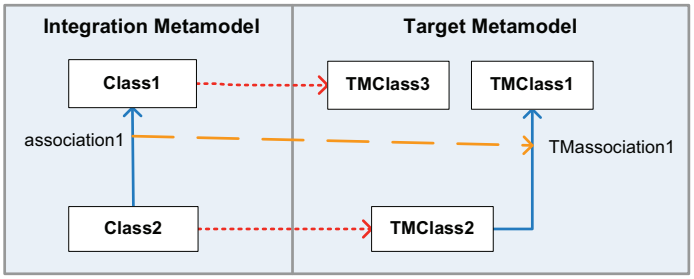

Figure 21. Example of an equivalent association with type difference

For the identification of differences in equivalent attributes, the data types and the enumerations must be considered. For the specification of data types, there are two alternatives:

1. The use of the primitive types that are implicit in the MOF specification, which are: Integer, Boolean, String, UnlimitedNatural.
2. The definition of new data types, which can be equivalent to primitive types or to specific data types of the target metamodel.

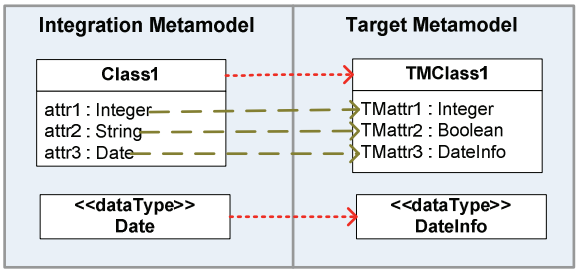

Figure 22. Example of mapping of equivalent attributes

Figure 22 shows an example Integration Metamodel that has the class *Class1*. This class has the three attributes *attr1*, *attr2*, *attr3*, which are equivalent to the attributes TMattr1, TMattr2, TMattr3 of the target metamodel. According to the mapping presented, for the attributes attr1 and TMattr1, there is not a type difference because both attributes are defined with the primitive type *Integer*. Between the attributes *attr2* and *TMattr2* there is a type difference because *attr2* has the primitive type *String* while *TMattr2* has

the primitive type *Boolean*. Finally, for the attributes attr3 and TMattr3, seems that there is a type difference because the name of the type related to *attr3* (*Date*) differs from the name of the type related to *TMattr2* (*DateInfo*). However, since the data type *Date* is equivalent to *DateInfo* according to the defined mapping, we consider that these two data types are equivalent, and, hence, there are not differences in the type of the two association ends involved.

For equivalent attributes that are specified by means of enumerations, the criteria that must be used for the identification of type differences is similar to the one applied to data types and associations (association ends). It is necessary to use the mapping information to determine if the enumerations that are related to source and target elements are equivalent. Figure 23 exemplifies this situation.

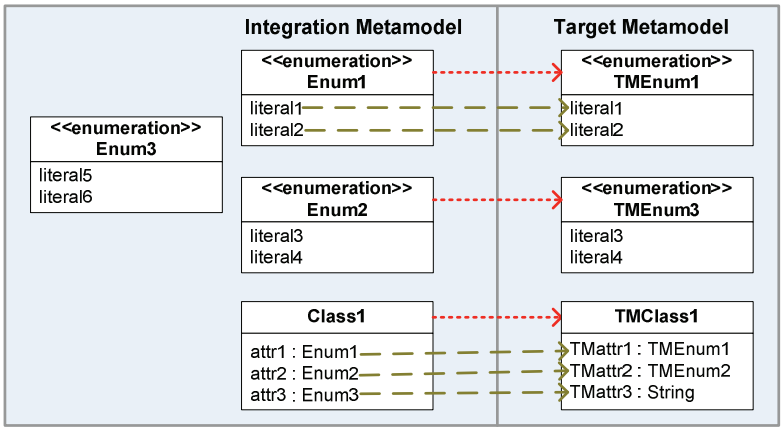

Figure 23. Example of enumeration mappings

In the Integration Metamodel that is presented in Figure 23, there are three different enumerations, which correspond to the types of the three attributes that are defined in the class *Class1*. In this figure it can be observed that there are not type differences for the equivalent attribute *attr1* because the enumerations related to the types of the source and target attributes are equivalent (*Enum1* is equivalent to *TMEnum1*). For the attribute *attr2*, which is equivalent to *TMattr2*, there is a type difference; the type of attr2 (*Enum2*) is not equivalent to the type of *TMattr2* (*TMEnum3*). For the equivalent attribute *attr3* the type difference is really clear since the type related to this attribute is an enumeration while the type related to the target attribute is a primitive type.

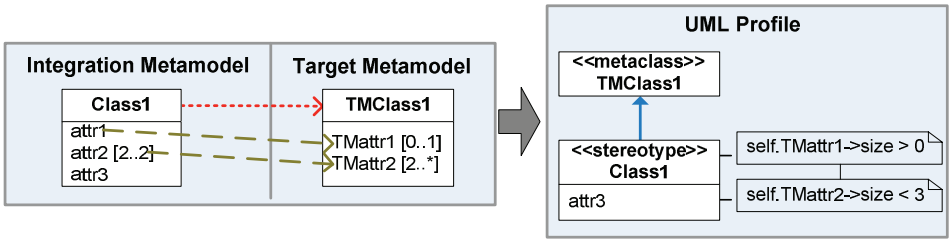

Figure 24. Example of extension identification from new attributes and cardinality differences

Figure 24 exemplifies how the differences that exist between the Integration Metamodel and the target metamodel allow the correct identification of the metamodel extensions that must be implemented in the final UML profile. In this figure, two kinds of differences are shown: new attributes, and cardinality differences in equivalent attributes.

Figure 24 shows an Integration Metamodel with a class named *Class1* that is mapped to the class *TMClass1* of the target metamodel. The attributes *attr1* and *attr2* of the Integration Metamodel are equivalent to the attributes *TMattr1* and *TMattr2* of the target metamodel. These two attributes present cardinality differences in relation to the mapped attributes of the target metamodel. Also, the attribute *attr3* of the Integration Metamodel has no equivalence in the target metamodel; therefore, we consider this attribute as a *new attribute*. With this mapping, the UML profile that is shown at the right of the figure is obtained. In the resultant UML profile, the stereotype *Class1* is defined to represent the corresponding class of the Integration Metamodel, which is extending the target class *TMClass1*.

In the stereotype that is defined in the UML profile, the cardinality difference that exist between *Class1.attr1* and *TMClass1.attr1* is solved by means of a OCL constraint that increases the lower bound cardinality of the attribute *TMattr1* of the target metamodel. A similar OCL rule is defined to constraint the difference of the lower bound cardinality of the attribute *TMattr2*. Finally the new attribute *attr3* is represented by means of the tagged value *attr3* in the stereotype *Class1*.

With these three extensions (the two OCL rules and the tagged value defined) the difference that exist between the Integration metamodel and the target metamodel are solved. This is, the abstract syntax that is represented in the Integration Metamodel has been integrated into the target metamodel.

If we apply the extension identification to the Integration Metamodel presented as example in Chapter V, the comparison results presented in Figure 25 are obtained. In this figure, the mappings and new elements that present differences are enumerated from 1 to 4. In the comparison results, the identified differences are shown by indicating (when necessary) the values for the Integration Metamodel element (*I.M.*) and the UML element (*UML*).

The mapping information defined in the Integration Metamodel allows the identification of type differences. For instance, in the case of *DMAssociationEnd.type* and *Property.type*, the type is different because *DMClass* is not equivalent to *Type*.

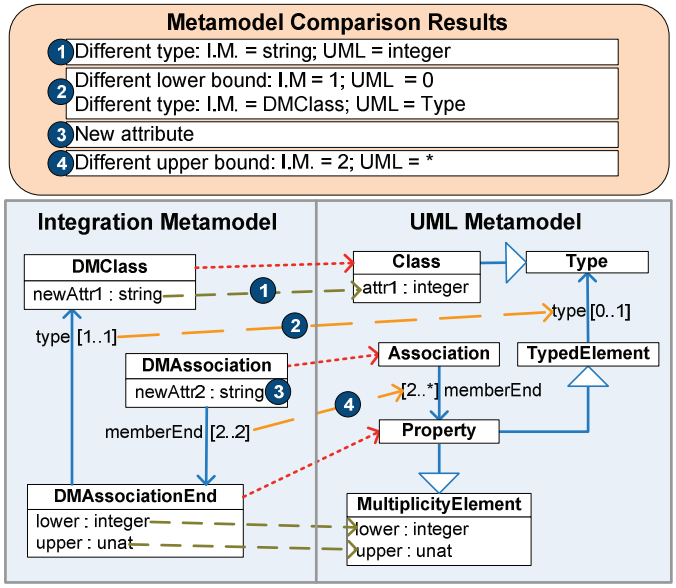

Figure 25. Metamodel Comparison Results

The cardinality differences are identified by analyzing the lower and upper bound of the equivalent properties and the referenced UML properties. This is the case of the equivalent properties *DMAssociation.memberEnd* and *DMAssociationEnd.type*. The differences identified in the comparison of metamodels indicate the extensions that must be introduced into the target metamodel in order to correctly represent the modeling needs of the related MDD approach. Thus, Table 4 shows the required extensions according to the results presented in Figure 25. In this table, the column *Target Element* shows the element of the target metamodel that must be extended (customized), and

column *Extension* shows the metamodel extension (customization) that is necessary to solve the differences identified.

Table 4. Metamodel Extensions Identified

Target Element	Extension
DMClass.newAttr1	Change attribute type from Integer to String
DMAssociation.memberEnd	Change the upper bound cardinality from many to 2
DMAssociation	Create the attribute *newAttr2* with type *String*
DMAssociationEnd.type	Change lower bound cardinality from 0 to 1 Change the association type from *Type* to *DMClass*

6.3. Integration Metamodel Transformation

The second step of the automatic UML profile generation defines a set of rules to automatically transform the Integration Metamodel and the metamodel extensions previously obtained in the corresponding UML profile. These transformation rules are defined considering that the *new elements* and the *extensions* identified during the metamodel comparison must be represented in the generated UML profile. Also, the transformation rules take into account the automatic generation of constraints to assure the correct application of the generated extensions.

The mapping information between the Integration Metamodel and the target metamodel extended with the generated UML profile is also automatically obtained during the Integration metamodel transformation. This mapping is essential to transform models defined with the extended modeling language (target modeling language) into equivalent models based on the source metamodel (the metamodel of the reference MDD approach).

In order to show how the Integration Metamodel can be transformed into the corresponding UML profile and the corresponding mapping is obtained, the required transformation rules are described below. These transformation rules are separated by the different EMOF conceptual constructs. The possible modeling situations are analyzed for each construct, according to the Integration Metamodel features. A figure that exemplifies the application of the transformation rules in a generic way is also presented.

6.3.1. Classes

Rule 1: One Stereotype for each *equivalent class*. The stereotype extends the referenced class of the target metamodel, and its name is equal to the *equivalent class* name. Figure 26 exemplifies this rule.

If the name of the *equivalent class* is equal to the name of the target class, then a prefix is used in the generated stereotype to differentiate the resultant stereotype and the extended class.

This first transformation rule is the most relevant because it involves the generation of the stereotypes, which are the main constructs of the UML profile. The rest of transformation rules are applied according to the results obtained by this first rule.

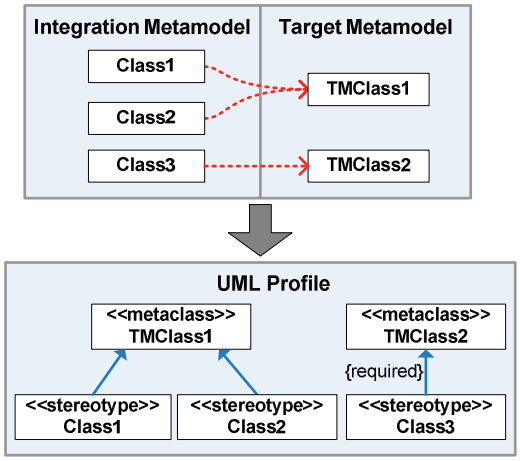

Figure 26. Generic transformation example for Rule 1

Constraint: At the end of the UML profile generation, if there is only one stereotype that extends a class of the target metamodel, then the stereotype extension must be defined as *required*. This constraint is defined because, in the DSML context, the target class only has the semantics of the involved equivalent class (see the transformation of *Class3* in Figure 26).

Mapping: The *equivalent class* will be mapped to the corresponding stereotype generated by the transformation rule (see Figure 27).

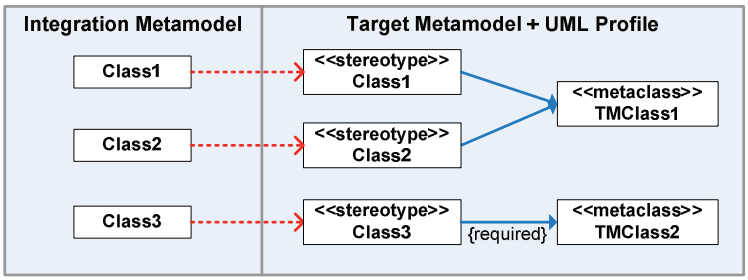

Figure 27. Mapping obtained for the example related to Rule 1

6.3.2. Properties

In EMOF, the properties represent attributes of a class (metaclass) or references (associations) between the classes. The main difference between an attribute and an association is that an attribute represents a data-valued property, while an association is an object-valued property. In other words, in an association, the type is given by another class of the model that represents the related class. These differences are taken into account in the definition of the involved transformation rules.

Rule 2: One tagged value for each *new property*. The tagged value must have the same type and cardinality as the *new property*. The name of the tagged value must be the name of the *new property*. In the case of an association, the tagged value must have the same aggregation kind as the *new property*. The application of this rule can be observed in Figure 28 for the association *Class1.rolClass2*.

Mapping: The *new property* of the Integration Metamodel is mapped to the tagged value generated in the UML profile (see Figure 29).

Rule 3: One OCL constraint if the lower bound of an *equivalent property* is higher than the lower bound of the referenced property:

self.*[property]*->size() >= *[newLowerBound]*

Rule 4: One OCL constraint if the upper bound of an *equivalent property* is lower than the upper bound of the referenced property:

self.[property]->size() <= [newUpperBound]

As Figure 28 shows, rules 3 and 4 are applied to the *Class2.roleClass3* and *Class3.roleClass1*, respectively.

Constraint: For rules 3 and 4, an OCL constraint is defined to validate that the corresponding stereotype is applied each time that the involved association is established. Thus, the type of the referenced association is restricted to the stereotype that represents the type of the *equivalent association*:

```
self.[equivalentAssociation]->isStereotyped*([newType])
```

This validation is also applied if the type of an *equivalent association* is changed by a specialization of the original type (see *Class2.rolClass3* in Figure 28).

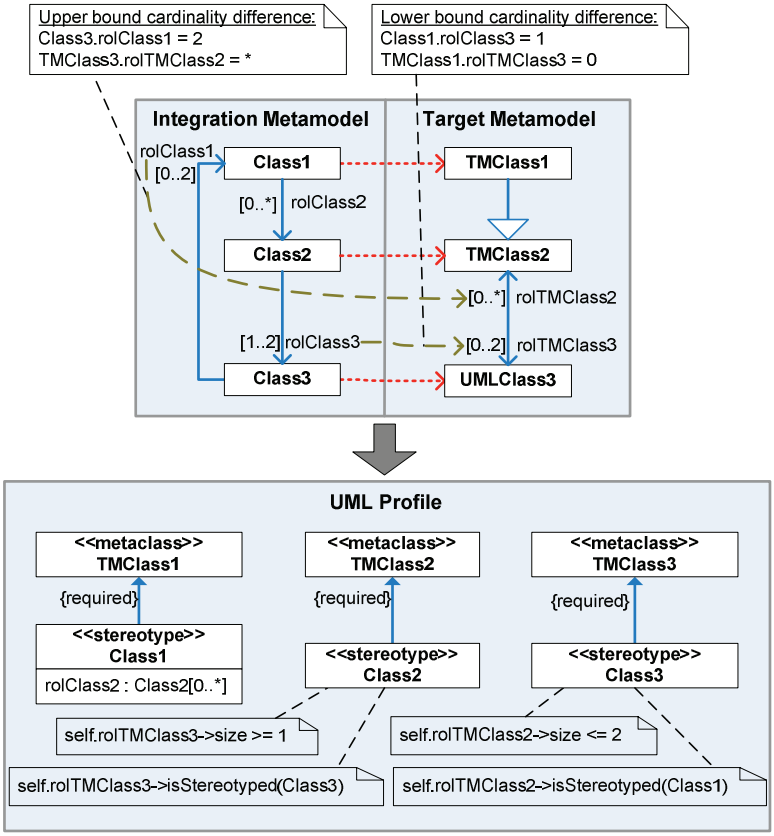

Figure 28. Generic example for the transformation rules 2 to 4

* The OCL operation *isStereotyped* is not part of the OMG specification and is only used to simplify the OCL rules presented. In the application of the integration process, this operation must be implemented according to the target UML tool.

Mapping: For rules 3 and 4, the *equivalent properties* of the Integration Metamodel are mapped to the corresponding properties of the target metamodel according to the original mapping defined for the Integration Metamodel and the target metamodel (see Figure 29).

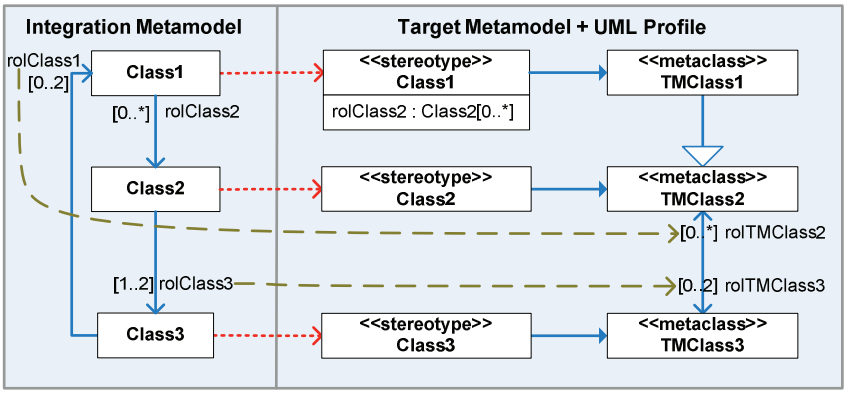

Figure 29. Mapping obtained for the transformation example related to rules 2 to 4

Even though an extension relationship represents a refinement of a class in a way similar to a generalization relationship, its semantics is represented as a special kind of association and not as a generalization. For this reason, a tagged value cannot redefine properties. Therefore, when the differences that exist between an *equivalent property* and the referenced property cannot be represented using OCL constraints, a tagged value that replaces the referenced property is created. In this case, the MDD process must only consider the new tagged value defined and not the property of the target metamodel that was originally referenced.

Rule 5: One tagged value that replaces a property of the target metamodel when one of the following conditions holds:

- The type of *equivalent property* is different than the type of the referenced property, and the new type is not a specialization of the original type or a stereotype that extends the original type (see *Class1.attr3* in Figure 30).
- The upper bound of the equivalent property is higher than the upper bound of the referenced property (see Class1.attr2 in Figure 30).
- The lower bound of the equivalent property is lower than the lower bound of the referenced property (see Class1.attr1 in Figure 30).

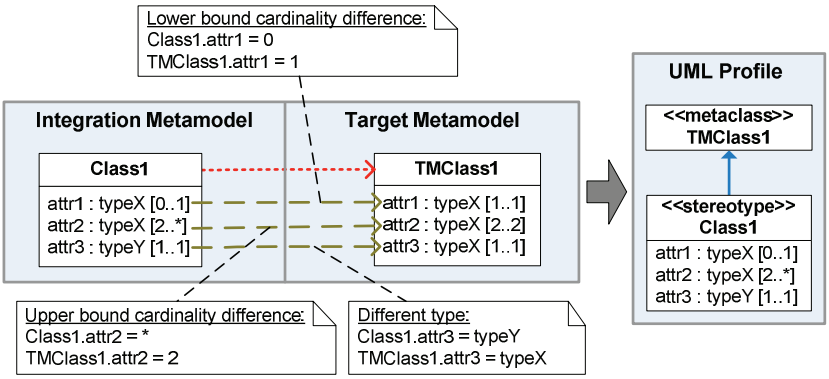

Figure 30. Generic example for transformation rule 5.

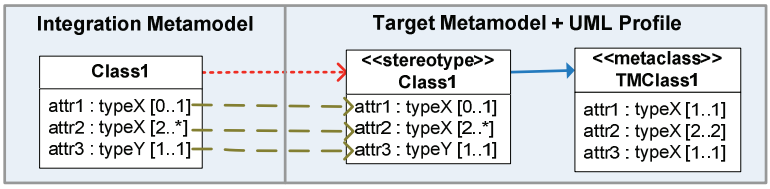

Figure 31. Mapping obtained for the transformation example related to Rule 5

6.3.3. Enumerations

The enumerations are used to specify a customized set of values that can be represented by an attribute of a class. Graphically, the enumerations are represented as a class. However, the enumeration is a specialization of a *Classifier* and not of a *Class*. Hence, an enumeration is not a class of the metamodel and it cannot be extended by a stereotype. This difference is considered in the following transformation rule.

Rule 6: One enumeration for each *new enumeration* or *equivalent enumeration* with new literal values. In the case of an *equivalent enumeration*, the generated enumeration replaces the original enumeration, and the involved *equivalent attributes* are considered as *new attributes* (Rule 2). This replacement is performed due to the referenced enumeration of the target metamodel cannot be extended with a stereotype in order to include the new literal values. Figure 32 shows the application of this rule for *Enum2* (*equivalent enumeration*) and *Enum3* (*new enumeration*).

Constraint: One OCL constraint for each attribute whose type corresponds to an *equivalent enumeration* that has fewer alternatives (literal values) than the referenced enumeration (see *Class1.attr1* and *Enum1* in Figure 32).

self.*[attribute]* <> #*[nonMappedLiteralValue]*

This constraint avoids the use of invalid alternatives (non-referenced literal values) that are defined in the referenced enumeration.

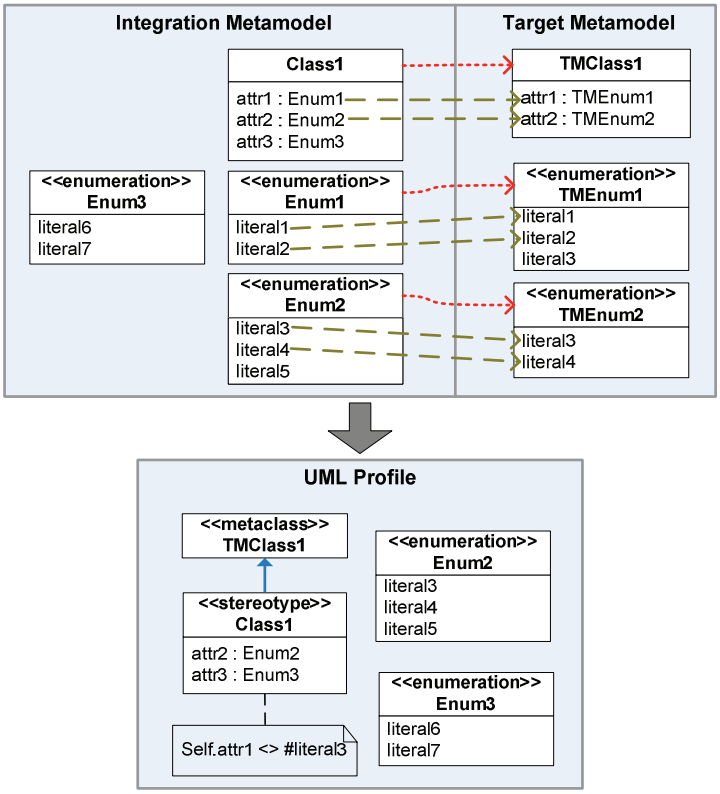

Figure 32. Generic example for transformation rule 6.

Mapping: The enumeration and corresponding literal values of the Integration Metamodel are mapped to the enumeration and literal values generated by the execution of the transformation rule. For equivalent enumerations that does not present new literal values, the resultant mapping is the same as the originally defined for the Integration Metamodel.

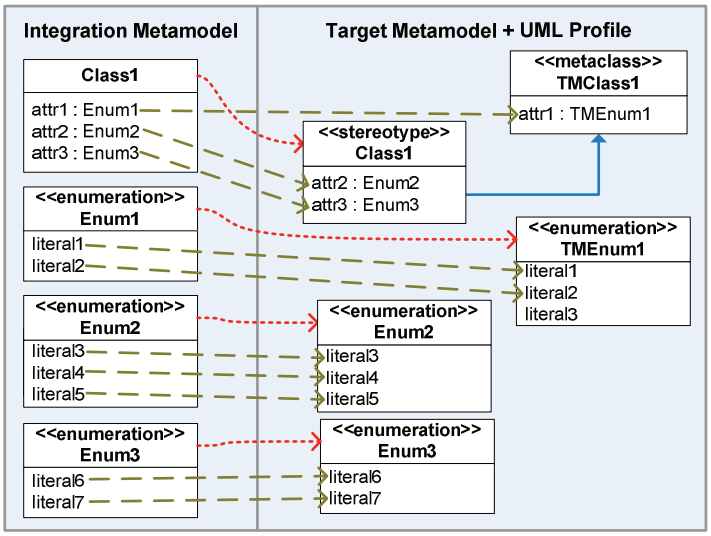

Figure 33. Mapping obtained for the transformation example related to Rule 6

6.3.4. Generalizations

The generalization relationships have interesting features that must be considered in the generation of the related stereotypes. Two of the main features that must be considered are the following:

1. Stereotypes are a special kind of class; therefore, it is possible to define a generalization between stereotypes
2. The extension association between a stereotype and its related class is a specialization of *Association*; therefore, the extension relationship can be inherited.

These two features are considered in the definition of the transformation rules related to the generalizations, which are presented below.

Rule 7: Define one generalization between two stereotypes that represent *equivalent classes* that are associated with a *new generalization* and that are referencing the same target class. The extension related to the child stereotype is not defined since it is implicit in the generalization relationship. Figure 34 shows the application of this rule for the generalization defined between the classes *Class1* and *Class3*.

Rule 8: If there is a *new generalization* between two *equivalent classes* that are referencing different classes of the target metamodel, the generalization relationship is not represented in the UML profile. In this case, the extensions of each stereotype to the corresponding class are defined, and the inherited properties (attributes and associations) are duplicated (see the generalization between classes *Class3* and *Class4* in Figure 34). If the generalization is represented, then the child stereotype will be able to extend the class of the target metamodel that is extended by the father stereotype. This could produce a modeling error since, according to the mapping information, the child stereotype is referencing (extends) a different class.

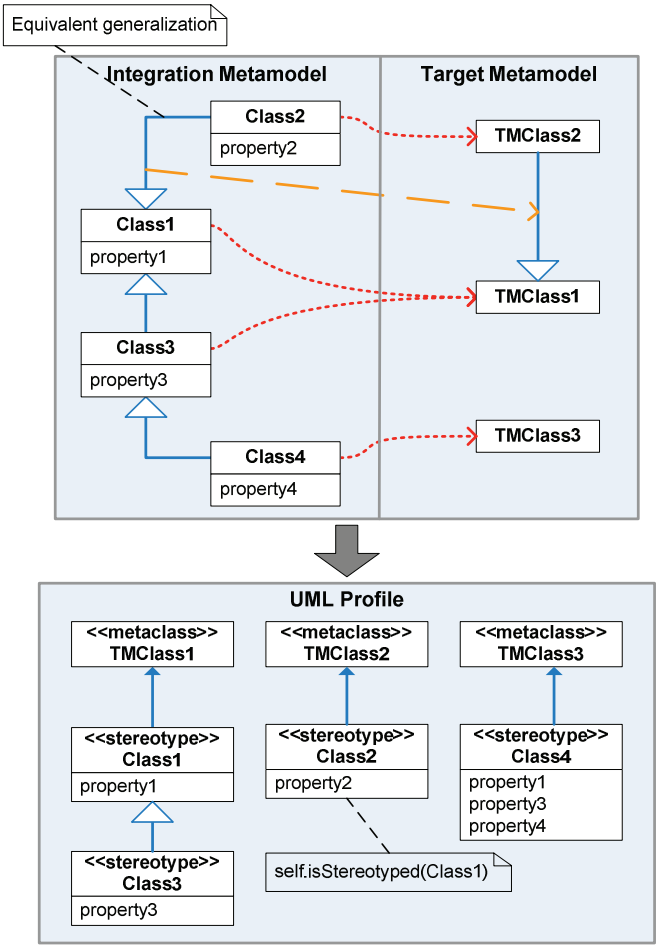

Figure 34. Generic example for transformation rules 7, 8 and 9.

Rule 9: If there is an *equivalent generalization* between two *equivalent classes*, the generalization relationship is not represented in the UML profile, and only the extensions of each stereotype are defined to the corresponding class of the target metamodel. In this way, the generalization defined in the target metamodel is used instead of the *equivalent generalization* (see the generalization between classes *Class1* and *Class2* in Figure 34).

Note that an *equivalent generalization* represents a generalization that already exists in the target metamodel. The *equivalent generalizations* are automatically identified through the participant classes of the Integration Metamodel that are equivalent to the classes that participate in the generalization of the target metamodel.

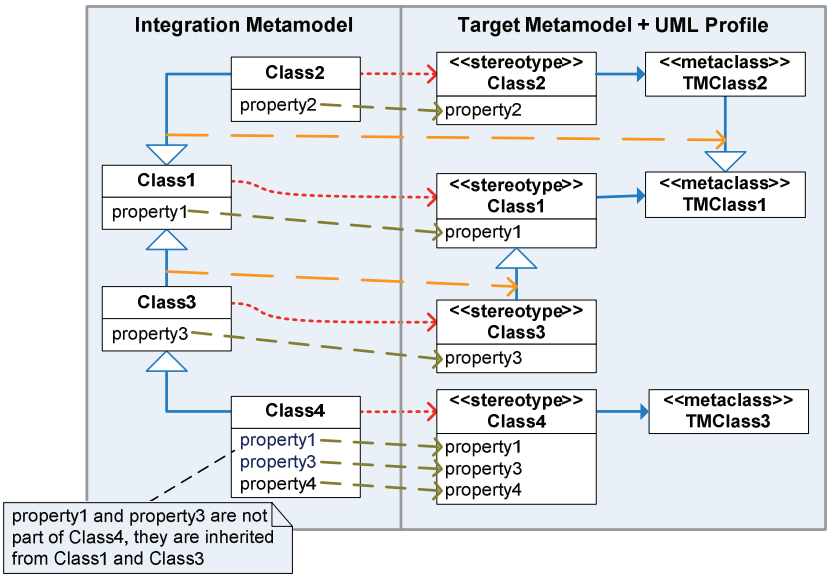

Figure 35. Mapping obtained for the transformation example related to rules 7 to 9

Mapping: After the application of the transformation rules 7 to 9, the generalization relationships of the Integration Metamodel are not mapped to the corresponding relationships in the target metamodel nor the generated UML profile. The equivalences that exist between the generalization in the Integration Metamodel and the extended target metamodel are automatically inferred. The mapping for classes and attributes must be defined according to the rules previously presented for classes and attributes according to the corresponding modeling situation. The only especial situation is related to the Rule 9 that

requires the definition of new attributes (tagged values) in the generated UML profile. In this case, it must be defined a mapping between the inherited properties of the related child class and the attributes generated by the transformation rule. Figure 35 shows the mapping obtained for the generic example presented in Figure 34 that is related to rules 7 to 9. Even though in Figure 35 are indicated the equivalences between generalization of the Integration Metamodel and the target metamodel extended with the generated UML profile, this is only to facilitate the understanding of the resultant mapping since, as mentioned before, this mapping is not really obtained from the UML profile generation.

6.3.5. OCL Rules

The OCL rules defined in the Integration Metamodel manage the interactions between the different constructs of the source DSML. Therefore, these rules must be included in the generated UML profile.

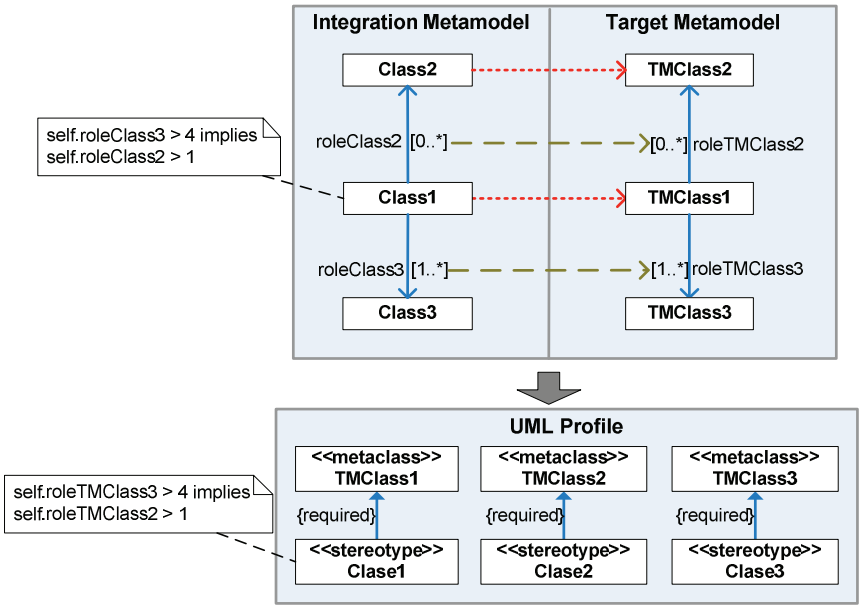

Figure 36. Generic example for Rule 10

Rule 10: The OCL rules defined in the classes of the Integration Metamodel must be included in the stereotypes generated from these classes. The elements referenced in the rules must be changed by the corresponding classes of the target metamodel and the stereotypes that were generated in the Integration Metamodel transformation process. Figure 36 shows an example of the application of this transformation rule.

Mapping: The application of the transformation rule number 10 does not imply the generation of new mapping information. However, it requires the mapping information that is generated from the other transformation rules for its correct application.

6.3.6. Data Types

Rule 11: The UML profile specification does not directly support the definition of data types; however, it is possible to import specific model elements in the UML profile by means of a model library. Thus, the *new data types* defined in the Integration Metamodel are defined in a separate model library that is imported in the UML profile generated.

The equivalent data types that have differences in relation to the referenced data types are considered as *new data types*. Since the data types are classifiers, they cannot be extended using stereotypes.

Rule 11 is the last rule defined for the transformation of the Integration Metamodel in the equivalent UML profile. Figure 37 exemplifies the application of this transformation rule in a generic example.

Mapping: The equivalent attributes that are related to equivalent data types that present differences must be mapped to the corresponding attributes (tagged values) defined in the UML profile. The data types of the integration metamodel do not require to be mapped to the target metamodel extended with the UML profile. The mapping obtained from the Integration Metamodel transformation assures that all mapped properties have a correct correspondence of type with the corresponding properties of the target metamodel and the UML profile generated. Figure 38 shows the mapping obtained for the transformation example presented in Figure 37.

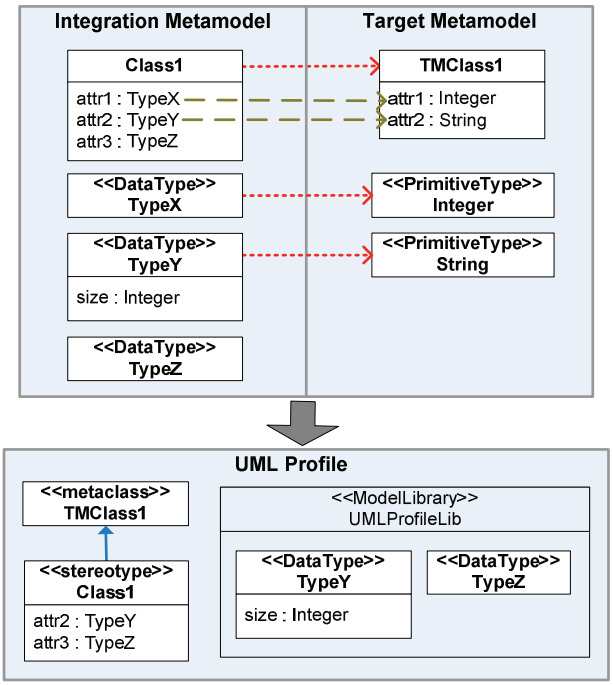

Figure 37. Generic example for Rule 11

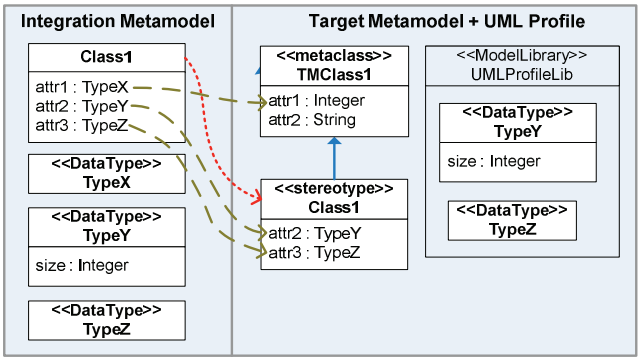

Figure 38. Mapping obtained for the example related to Rule 11

6.4. Applying the Transformation Rules

The eleven transformation rules that are presented above allow the automatic transformation of an Integration Metamodel into the corresponding UML profile.

Figure 39 presents the UML profile obtained after applying the proposed transformation rules to the example Integration Metamodel and the results obtained from the comparison between the example Integration metamodel and the target metamodel (UML metamodel in the example).

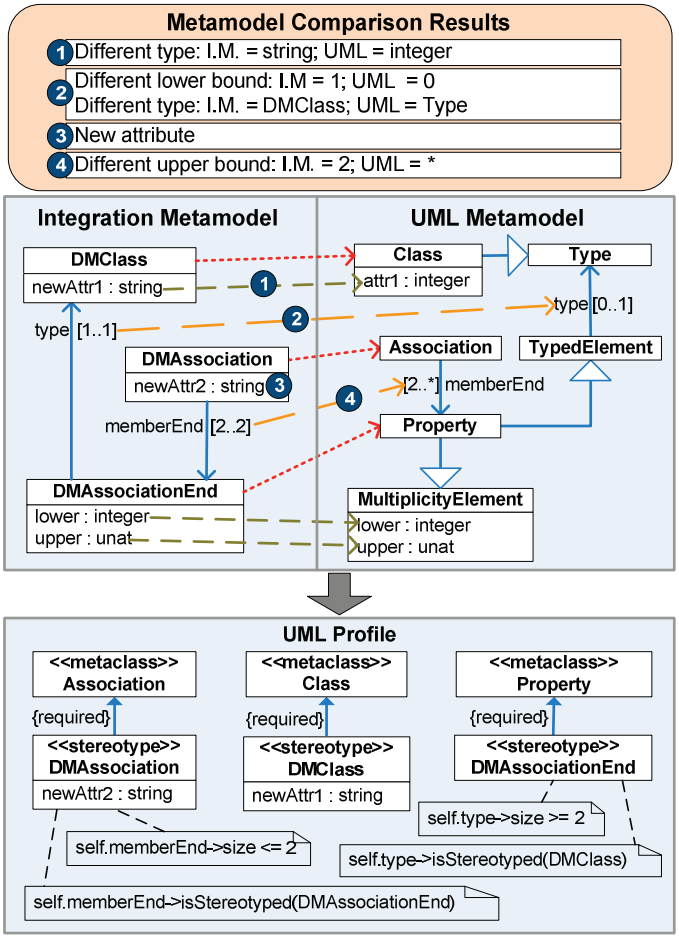

Figure 39. UML profile generated for the example Integration Metamodel

In addition to the UML profile, the transformation of the Integration Metamodel also generates new mapping information that takes into account the generated UML profile elements (stereotypes, tagged values, etc.). This new mapping provides the equivalence between the Integration Metamodel and the UML Metamodel (target metamodel) extended with the generated UML profile. Figure 40 shows this new mapping information for the UML profile presented

in Figure 39. According to the obtained mapping, all the constructs and properties that are present in the Integration Metamodel (that represents the source DSML) have a direct correspondence in the extended metamodel (the UML metamodel).

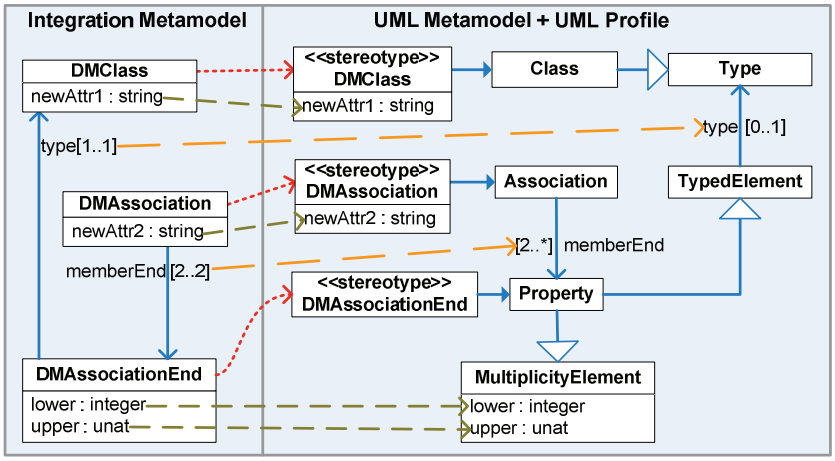

Figure 40. Mapping generated for the UML profile presented in Figure 39

The generated UML profile together with the obtained mapping definition can be used to interchange extended models (that are related to the target metamodel) and DSML models (that are related to the source metamodel) [136]. In the proposed example, this model interchange can be used to take advantage of the modeling benefits (tools, knowledge, notation, etc.) that a standard modeling language such as UML provides to support a specific MDD process.

6.5. Conclusions

This chapter presents a solution for the automatic generation of a UML profile from the metamodel that represents the DSML related to a MDD approach.

The proposed solution tackles an important topic that has not yet received the required attention: the correct definition of UML profiles for MDD solutions. Even though the number of UML profile solutions has increased, the number of publications related to a correct UML profile definition is very limited [17] (see Background in Chapter II). In order to obtain this correct definition, the

proposed transformations are focused on three main elements. These elements are the following:

1. The generation of those modeling elements defined in the source DSML that are not present in the target modeling language. This is oriented to obtain a sound representation of the models required by a specific MDD approach by using an existing modeling approach that is customized with the generated UML profile.
2. The management of differences that could exist between elements of the source DSML that are equivalent with elements of the target modeling language. This aspect is very relevant, since not only is necessary to extend/customize the target modeling languages with the additional features that are present in the proprietary DSML, but also, it is important to manage those differences between the participant modeling languages that could prevent an appropriate definition of the required models.
3. The generation of constraints to assure that the application of the generated UML profile follows the DSML specification. This is a very important point that is not considered by most of the current UML profile proposals. The definition of the structural features that are required by the MDD approach in the generated UML profile do not guarantee the correct generation of models with the customized modeling language. It is also important to assure that the defined extensions are well applied to obtain properly defined models in the context of the MDD approach. This is achieved by means the generation of specific constraints for each extension generated.

It is important to note that the transformation rules that are defined in this chapter are just one possible solution for the complete generation of a correct UML profile. Variations of these transformation rules can be defined depending on different design decisions.

The following chapter explains how the interchange of models is performed by using the artifacts obtained from the Integration Metamodel definition and the UML profile generation presented in this chapter.

Chapter VII.
Generation of Model Interchange Mechanisms

Nowadays, the modeling approaches related to MDD proposals are considered isolated modeling alternatives. These are not developed to interchange information with other modeling approaches, which prevent their re-use in different MDD processes or their application to different development scenarios (with different modeling abstraction levels). In this chapter, we show how this MDD reality can change, by means of a proposal that generates model-to-model transformation mechanisms to automatically interchange information among different modeling approaches. The generation of these transformation mechanisms is performed by means of the artifacts generated from the first three steps of the proposed MDD interoperability process. Thus, the proposal presented in this chapter shows how proprietary DSMLs and the customized modeling languages can interoperate to provide benefits for the application of MDD solutions.

One of the most important concerns when elaborating a *Model-Driven Development* (MDD) solution [1] is the specification of a modeling language that allows the required software products to be represented at the conceptual level without ambiguity. Among the different choices that exist for the definition of an adequate modeling language, there are two alternatives that appear to be the most suitable. The first of these is the creation of a *Domain-Specific Modeling Language* (DSML) [3] that is tailor-made for the MDD approach. . The second alternative is the customization of previously-existing modeling languages by means of extensions defined in the corresponding metamodels, which represent the abstract syntax related to the semantics required for the MDD proposal [17]. In this thesis, we have considered the definition of these extensions by means of the standard metamodel extension mechanisms that is proposed by OMG [34], the UML profile [51].

This chapter presents the proposal defined to automatically generate mechanisms for the interchange of models related to proprietary DSMLs and customized modeling languages. Thus, it would be possible to take advantage of the existing MDD tools for those models that can be represented by customized modeling languages and only to implement new tools for those MDD tasks that are related to specific aspects of the MDD approach, such as model compilers. It would also be possible to implement specific modeling tools for those features that are outside of the scope of existing tools.

The interchange proposal is based on the definition of an Integration Metamodel and the automatic generation of a UML profile from the defined Integration Metamodel, which correspond to the steps 2 and 3 of the proposed MDD interoperability process (these steps are detailed in Chapter V and Chapter VI of this thesis). All the information required to generate the interchange mechanisms is obtained from these two steps of the process. For explanation purposes, the proposed interchange approach is applied to the same example presented in previous chapters, which is related to the customization of the UML association according to a binary relationship defined in a simplified DSML metamodel. Finally, a brief model compilation example is presented to show the application of this interchange proposal by means of the application of a hybrid modeling schema that integrates OO-Method and UML tools.

The rest of this chapter is organized as follows: Section 7.1 presents the background related to UML profiles and DSMLs. Section 7.2 presents our proposal for model interchange. Section 7.3 presents how to apply the

interchange proposal. Section 7.4 presents a brief discussion about the proposal, and finally, Section 7.5 presents our conclusions.

7.1. Background

This section briefly introduces the aspects that are relevant to the definition of DSMLs and UML profiles. This background is specially focused on those aspects that can be used to perform an interchange between proprietary DSML models and models that are customized with UML profiles.

7.1.1. Domain-Specific Modeling Languages

A *Domain-Specific Modeling Language* (DSML) represents the semantics of the constructs required for the definition of the conceptual models involved in a MDD solution. For the construction of a DSML, a common strategy is to define a metamodel to represent the abstract syntax of the required conceptual constructs [48]. For the elaboration of this *DSML Metamodel*, one of the most interesting alternatives is the use of the *Essential Meta Object Facility* (EMOF) standard defined by OMG [34]. EMOF is an essential set of metamodeling constructs, which is defined within the *Meta Object Facility* (MOF) specification [38]. In the context of this work, the use of EMOF, instead of the Complete MOF specification (CMOF or simply MOF), takes special relevance because the metamodeling capabilities provided by EMOF are very close to the extension capabilities that the UML profiles provide. Therefore, by using EMOF, the features of the resultant DSML metamodels can be represented as UML profiles extensions. By contrast, the complete MOF specification provides a set of metamodeling capabilities that cannot be expressed as UML profiles extensions. An example of this is that in MOF-based metamodel is possible the redefinition of properties, which is not part of the EMOF specification. In the context metamodeling extensions [33], the properties redefinition corresponds to a heavy-weight extension mechanism because implies a change in the extended metamodel: a property of the target metamodel is redefined (changed) by a property of the defined metamodel extension. However, since the UML profile is a light-weight extension mechanism, it cannot change the extended metamodel, and therefore, the properties defined as UML profile extensions cannot redefine properties that already exist in the target metamodel.

Another benefit of using EMOF is that it has a standardized XMI definition [40]. The benefit of using an XMI standardized definition for EMOF metamodels is that it facilitates the interchange and validation of the defined metamodels and support from existing metamodeling tools, such as *Eclipse Modeling Framework* (EMF) project [57], can be obtained. EMF is an Eclipse project that provides an open-source implementation of the EMOF standard, which is called *Ecore*. In addition, by means of tools like Eclipse GMF [50] and Eclipse ATL [58, 80], specific model editors and model transformations can be defined over Ecore metamodels [57].

7.1.2. UML Profiles

The UML profile extension mechanism is part of the UML specification and it is defined inside of the *UML Infrastructure* [35]. It defines the mechanisms used to adapt existing MOF-based metamodels to specific platforms, domains, business objects, or software process modeling. Since this extension mechanism is a part of the UML standard, it can be supported by UML tools. Additionally, the UML profile definitions and can be interchanged by means XMI specification. These features are relevant advantages of the UML profile over other metamodel customization mechanisms, which are have not a standard specification and interchange format. Hence, other extension mechanisms proposed are not supported by generic modeling tools (such as UML modeling tools) and cannot interoperate with different MDD technologies.

Generally speaking, UML profiles are manually elaborated without a well-defined process. This situation is motivated by the lack of a standard that specifies how the UML extensions must be defined [31]. For this reason, many of the existing UML profiles are invalid or of poor quality [17]. To avoid this situation, some works propose a more methodical solution that consists in the definition of a UML profile from the metamodel that describes the conceptual constructs required by MDD approaches. In other words, the UML profile is generated from a DSML metamodel [17, 51, 54]. This UML profile generation schema is based on the identification of the equivalences (correspondences) that exist between the source DSML and the target modeling language. This identification of equivalences is performed by means of a mapping between the different elements (classes, association, attributes, etc.) of the source DSML metamodel and the corresponding elements of the metamodel related to the target modeling language. Later, the identified equivalences are used to guide the

correct definition of the required extension over the target metamodel through the UML profile implementation.

Certain proposals state that these equivalences can be used to partially automate the UML profile generation [19, 54]. However, these proposals cannot provide a totally automated solution for the generation of a complete UML profile. This occurs because, in real MDD approaches, certain structural differences between the source DSML metamodel and the target metamodel may appear. This prevents the automated identification of all the extensions that must be performed in the target metamodel.

The Integration Metamodel proposal presented in Chapter V, which corresponds to the second step of the MDD interoperability process proposed in this thesis, defines a solution to solve these structural differences in order to obtain an adequate input for an automated UML profile generation. In addition, considering that the UML profile is generated from the DSML metamodel, during the generation of the UML profile also can be obtained the information of the equivalences (mapping) between the extended metamodel (target metamodel extended with the generated UML profile) and the source DSML metamodel. With this mapping information, models that are defined using the generated UML profile can be automatically transformed into the equivalent models related to the source metamodel, and vice versa. The interchange proposal presented in this chapter is based on this idea.

7.2. Model Interchange Proposal

Our interchange proposal is based on the outputs obtained from the application of the *Step 2* and *Step 3* of the MDD interoperability process proposed in this thesis, which correspond to the *Integration Metamodel Definition*, and the *Automatic UML Profile Generation*. In particular, the core elements for the automatic generation of model-to-model transformation are the Integration Metamodel, the generated UML profile, and the resultant mapping information. Thus, the generated transformation rules can automatically transform an instance of the target metamodel (extended/customized metamodel) into an equivalent instance of the source DSML metamodel and vice versa. In order to understand how the model-to-model interchange is performed, the same example used in

previous chapters will be used, which corresponds to a simplified version of a binary association and its integration into the UML metamodel.

For this example, the corresponding Integration Metamodel and mapping are defined (*Step 2* of the interoperability process). This special metamodel allows the abstract syntax represented in a source metamodel to be automatically integrated into a target metamodel by means of an automatic UML profile generation process (Step 3 of the interoperability process). Additionally, from the application of the UML profile generation process, a specific mapping between the Integration Metamodel and the UML metamodel extended with the generated UML profile is also generated.

From the first mapping obtained, the mapping between the source metamodel and the Integration Metamodel, a bidirectional correspondence is obtained between the involved metamodels. Therefore it is possible to know how an element of the Integration Metamodel can be transformed in the corresponding element(s) of the source metamodel and vice versa. The same occurs for the mapping automatically obtained in the UML profile generation. This is the mapping between the Integration Metamodel and the customized target metamodel.

Thus, we obtain that our interchange proposal is based on two model transformations (see Figure 41), which are performed using the two mappings obtained. These model transformations require the metamodel extensions that are implemented in the generated UML profile to be executed. The model transformations use the Integration Metamodel as a pivot metamodel that is used to go from an instance of the source metamodel to an instance of the target metamodel and vice versa.

For instance, to transform an instance of the target metamodel extended with the generated UML profile into an instance of the source metamodel, the mapping obtained from the UML profile generation is used to generate an instance of the Integration Metamodel (*intermediate model*). Next, from this *intermediate model*, a new transformation is performed by applying the mapping obtained from the Integration Metamodel definition. With this last transformation, an instance of the source metamodel that is equivalent to the input instance of the target metamodel is obtained. Figure 41 shows the general schema of the interchange proposal.

Figure 42 exemplifies this interchange proposal with a brief model-to-model transformation that is related to the metamodels and mappings obtained from the

example related to the integration of a binary association and the UML metamodel (presented in the previous chapters). This example represents an association *many-to-many* between the classes *Passenger* and *Flight*. The models that participate in this example are the UML model that is extended with the generated UML profile (in the upper side of the figure), the intermediate model (in the middle of the image), and the DSML model (in the lower side of the image).

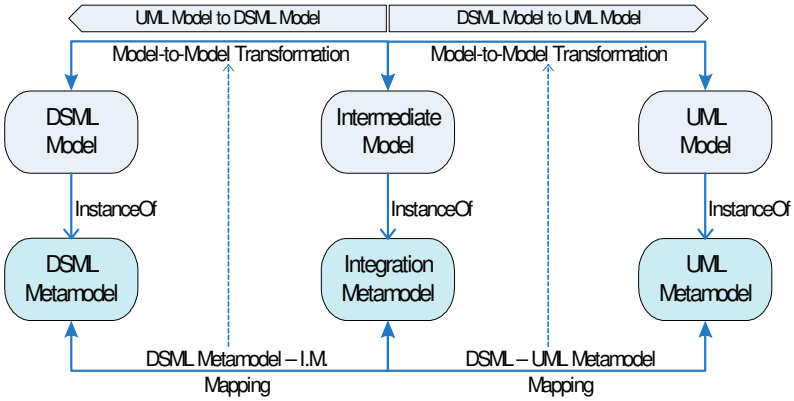

Figure 41. Schema of the Modeling Languages Interchange Proposal

It is important to point that the aim of the interchange proposal is to guarantee the transformation of an instance of the target metamodel into an instance of the source metamodel without losing modeling information. In this example this means to transform an instance of the target UML metamodel into an instance of the source DSML metamodel. Additionally, as we can observe in Figure 41, the interchange is based on bidirectional mappings, and, hence, it is also possible to perform the opposite transformation, which is, the transformation of an instance of the source DSML metamodel into an equivalent instance of the UML metamodel.

However, it is probable that the target metamodel be a more general language (in relation to the source metamodel) with a greater number of constructs. In fact, this is what we propose as basic for the application of the interoperability approach. Therefore, it is very possible that the model generated from the transformation of an instance of the source metamodel be an incomplete instance of the target metamodel, which must be refined to obtain an adequate (complete) instance of the target metamodel. This situation can be observed in the presented example, in particular, in the name of the association. According to the source

metamodel, the name of the association is not a relevant property. It is neither considered in the mappings nor in the interchange process. However, the association in UML requires an appropriate identification, which is performed by its name. Thus, a transformation of an instance of the source metamodel, using the mapping presented in the example, requires a refinement to specify the name related to each generated association.

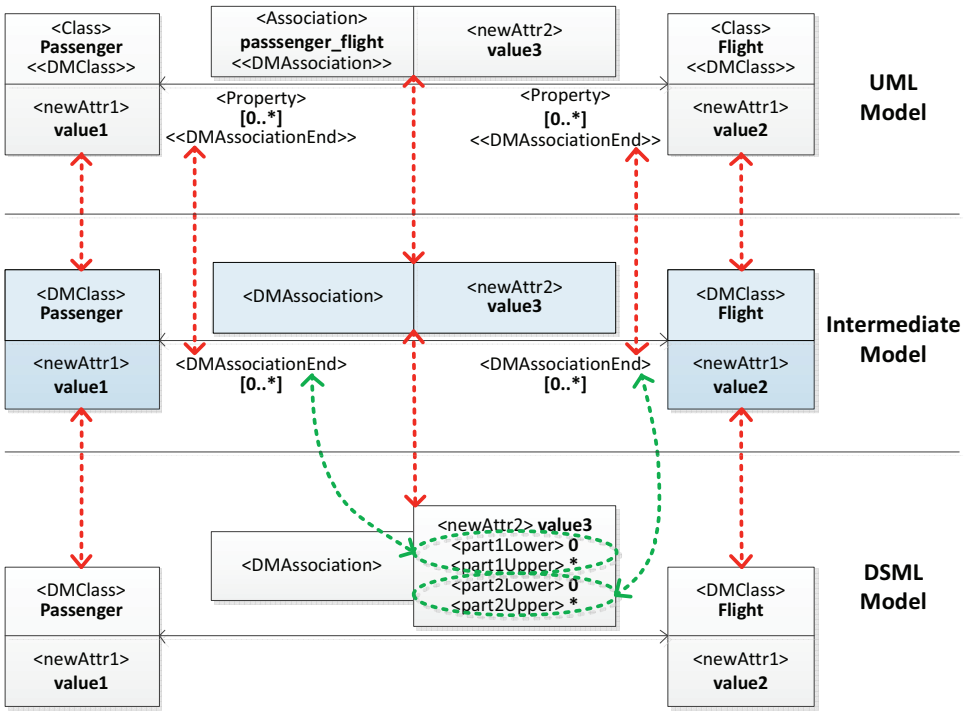

Figure 42. Interchange proposal application example

A recommended strategy for the implementation of the required model transformations is the use of model-to-model transformation technologies such as QVT or ATL. Thus, an interesting open-source implementation alternative is the Eclipse ATL project, which performs model transformations by means of a mapping that is defined between Ecore metamodels (EMF Project)[57]. In addition, the Eclipse ATL project also provides support to perform transformations over metamodels defined with the UML2 tools, which are part of the Eclipse UML2 project [126].

The proposal of Abouzahra et al. presented in [130] shows a practical approach based on ATL that can be useful to learn how transform a model

extended with a UML profile into another target model. To perform this transformation, the manual definition of a mapping between the UML profile and the metamodel related to the target model is required. In our proposal, this mapping corresponds to the mapping between the UML profile and the Integration Metamodel, which is automatically obtained. Thus, the Abouzahra et al. proposal could be an interesting open-source solution to implement the transformations between the UML model and the intermediate model that is presented in the interchange proposal. However, it is still necessary to implement the transformations to obtain the DSML model from the intermediate model, and vice versa.

Nevertheless, it is important to remark that for the implementation of the interchange proposal can be used other transformation alternatives, which are different than model-to-model transformations, for instance, by means of XSLT transformations.

The next section shows how has been applied the interchange proposal in the industrial implementation of the OO-method approach, and presents a brief example of its application.

7.3. Applying the Interchange Proposal

The proposed interchange proposal has been applied to an industrial MDD solution that is called *OlivaNova* (*the Programming Machine*) [62], which has been developed by CARE Technologies [64]. Olivanova corresponds to the industrial implementation of the OO-Method approach [12, 16]. Olivanova has been selected because it already has a suite of MDD tools that are based on the OO-Method DSML. Thus, by applying our interchange proposal to Olivanova, UML-based tools can be integrated with the existing Olivanova technology.

In the suite of Olivanova tools, there are two tools that are oriented to interchange UML and OO-Method models. These tools are the OO-Method XMI importer, which transforms an UML model into an OO-Method model, and the OO-Method XMI exporter, which transforms an OO-Method model into an UML model. The transformations performed by these tools are implemented through XSLT transformations.

It is important to mention that these two interchange tools correspond to the initial effort to obtain an interoperability mechanism for MMD approaches and UML, which is the starting point for developing the interoperability approach presented in this thesis.

However, the interchange provided by these interchange tools is limited by the UML modeling capabilities because the UML constructs do not provide all the precision required by the OO-Method conceptual model. In order to solve this problem, a UML profile that extends UML with the precision required by OO-Method has been defined using the first three steps of the interoperability process proposed. The mapping information obtained during the generation of the UML profile has been used to extend the OO-Method interchange tools in order to improve the interchange between UML and OO-Method models. In order to perform this improvement, the transformations required by the interchange proposal are implemented by means of XSLT transformations. This implementation decision has been taken because the OO-Method interchange tools are originally based on XSLT. Thus, it is not necessary to implement a completely new interchange tools, and only the new interchange aspects are included in the current implementation. This also demonstrates the platform independency that the proposed interchange approach has since it not only can be implemented by using tools based on model-to-model transformation technologies (such as ATL or QVT).

The platform independency of the proposed interchange approach is a very important point if we consider that the objective is to achieve the interoperation of different modeling language, which can have tool supporting implemented with different platforms and technologies.

The implementation schema that is used to apply the interchange proposal in the Olivanova technology is presented in Figure 43. According to this schema, the OO-Method metamodel is used as input for the UML profile generation process.

Figure 43 shows that the UML profile generation process also generates the mappings required to perform the model transformation involved in the interchange of UML and OO-Method models. It is important to note that the mapping between the Integration Metamodel and the UML metamodel is automatically generated during the automatic UML profile generation process.

In the importation process, the XMI importer tool uses as input the XMI definition of the UML model (extended with the generated UML profile) and

generates as output an equivalent OO-Method model, which is represented in XML format according to a specific DTD. The exportation process performs the opposite transformation, it takes as input the XML representation of an OO-Method model and generates as output an equivalent UML model. Therefore, the XMI definition of the UML model can be used by UML-based tools, and the XML definition of the OO-Method model can be used by the different Olivanova tools and by the Olivanova model compilers.

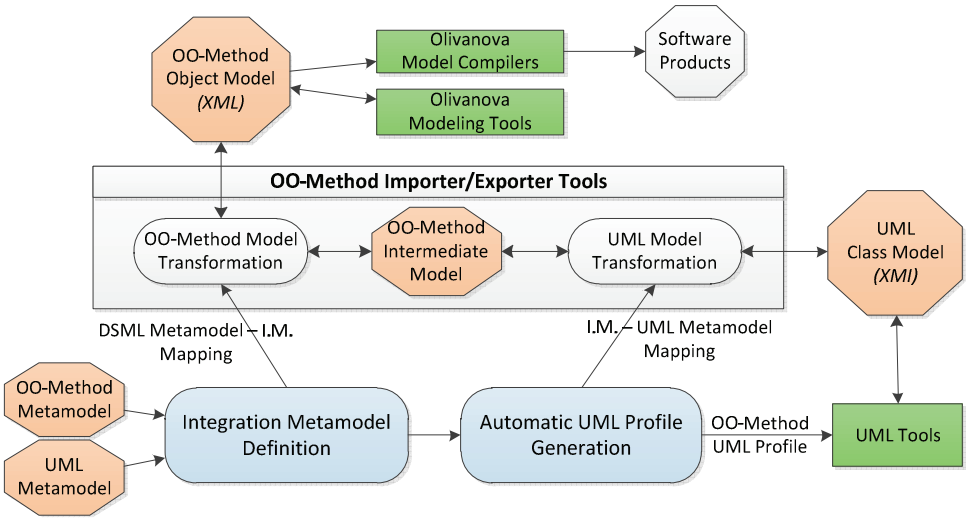

Figure 43. Schema for the implementation of the interchange proposal in OO-Method and UML

The interchange between OO-Method and UML is centered on the OO-Method object model, which is similar to the UML class model. Therefore, the interchange tools are focused on transforming an OO-Method object model into an UML class model, and vice versa. However, the OO-Method object model is only one of the diagrams involved in the definition of the OO-Method conceptual model, which is used by the Olivanova technology to automatically generate software applications throw a MDD process. There are other diagrams involved in the definition of the OO-Method conceptual model, for instance, the *Presentation Model*, which allows the user interface of the generated applications to be defined.

The object model has been selected for the interchange between OO-Method and UML because it is the core diagram for the definition of the OO-Method conceptual model. In addition, due to its proximity with the UML class model, it

could be more intuitive for defining the OO-Method object model using UML tools for those customers that already have experience in UML.

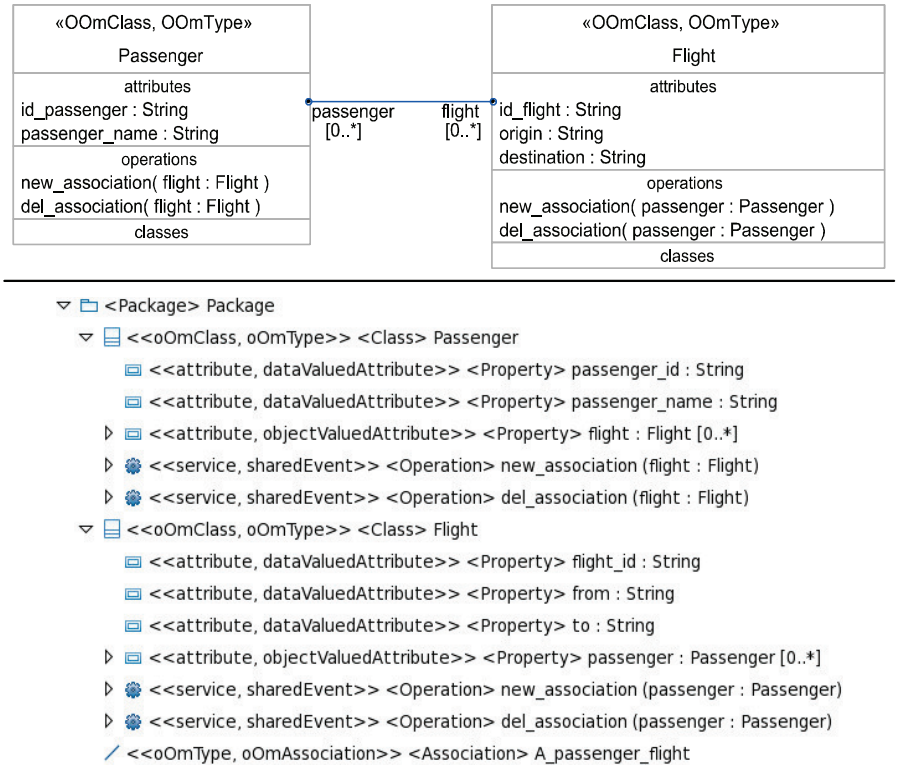

Figure 44. UML model extended with the OO-Method UML profile

Figure 44 shows a UML model that has been defined using the UML profile generated to represent the concepts of the OO-Method association. The generation of this UML profile is detailed in the next chapter, where the integration of UML and OO-Method is presented. The UML model describes the same example presented in Figure 42 (an association between the classes *Passenger* and *Flight*). Figure 44 also shows the description of the UML model in a tree view, where the application of the different stereotypes can be observed.

It is important to note that even though the OO-Method association is a binary association as the example presented in the UML profile generation

process, the OO-Method association requires additional properties that are not defined in that simplified example.

The UML model presented in Figure 44 has been specified using the Eclipse UML2 tool. This figure shows some of the OO-Method concepts that do not exist in UML, for instance, the concept of shared event that is represented by the stereotype *sharedEvent*. The shared events are services that are defined to manage the creation or destruction of links between instances of associated classes. Figure 44 also shows that the representation of the OO-Method class is performed by the stereotype *oOmClass*.

Later, this UML model is transformed into an equivalent OO-Method object model by means of the XMI importer tool (see Figure 45).

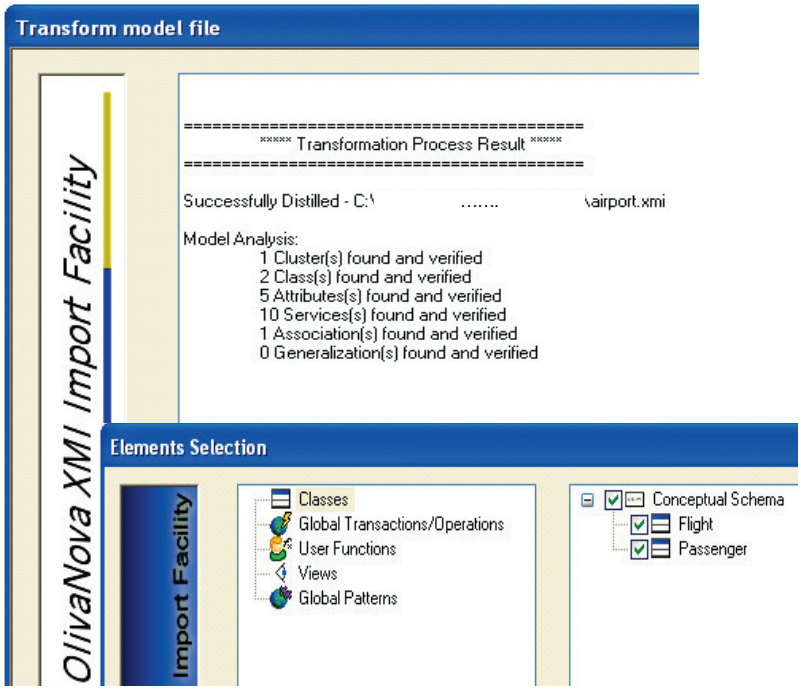

Figure 45. XMI importer application example

The XMI importer tool uses the mapping information obtained in the generation of the OO-Method profile (see Figure 43) to perform the transformations presented in the interchange proposal (see Figure 41).

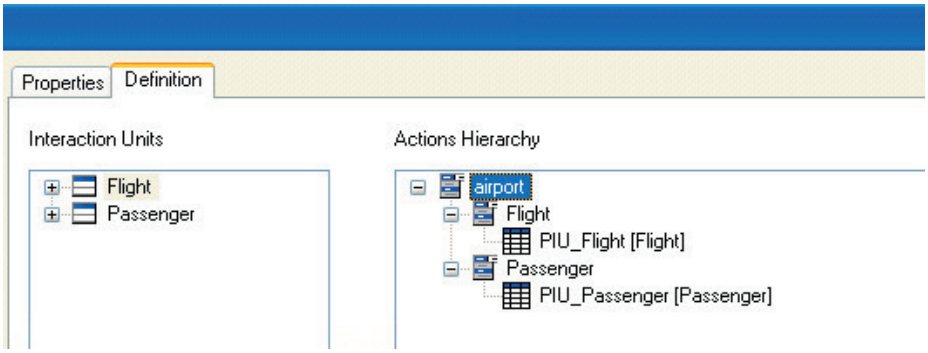

Figure 46. OO-Method presentation model

The presentation model can be defined from the obtained OO-Method object model using the OO-Method presentation model editor, which is based on the OO-Method DSML. Figure 46 shows a screenshot of this tool, which presents a partial view of the presentation model related to the imported UML model. This figure shows that the executable application will have two *Population Interaction Units* (PIU) to represent the instances related to each class of the model. A PIU represents an entry-point for the application, through the presentation of a set of instances of a class. An instance can be selected, and the corresponding set of actions and/or navigations specified in the presentation model are offered to the user. More details can be found in [12].

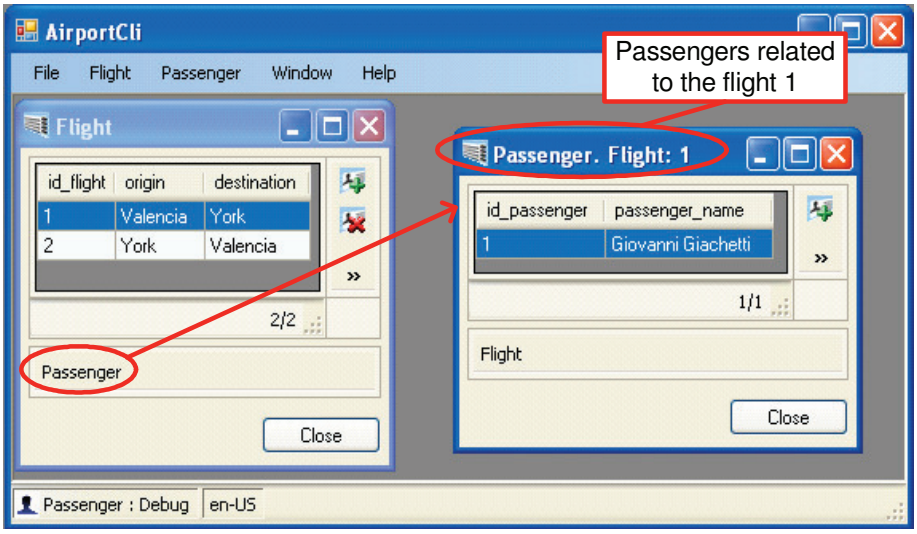

Figure 47. Generated application

Figure 47 shows a screenshot of the application that is generated from the imported UML class model and the defined OO-Method presentation model. This figure also shows that the association between the classes (defined in the UML model) is used to implement the navigation between related instances.

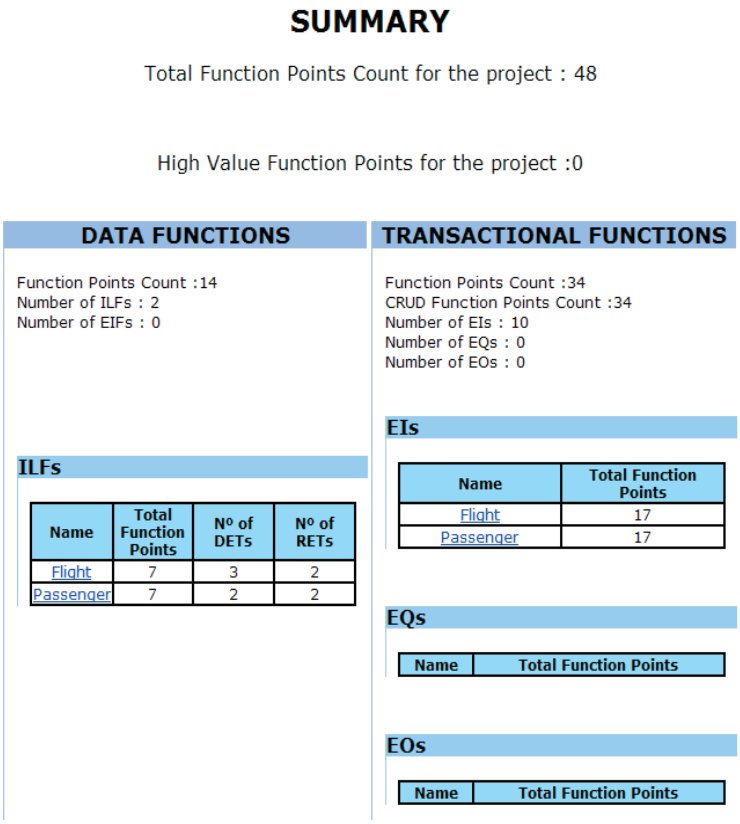

Figure 48. Functional size report generated for the application generated from the example UML model

Finally, the imported UML model can also be used to obtain other software products; for instance, to obtain the functional size of the modeled application with the OO-Method function points tools [137-139] in order to estimate the cost of the generated application. Figure 48 shows a general view of the report of function points that has been automatically generated from the example UML model. This report presents the functional size obtained by a measurement procedure defined according to the IFPUG FPA standard [140].

7.4. Comparing the Model Interchange Approach

Nowadays, the use of model transformations directed by metamodels mappings seems to be the most appropriate way for the interchange of models that are related to model-based technologies. In this context, certain works have proposed alternatives to facilitate the definition of the required mappings and transformations, such as [22]. This work presents a proposal to perform an automatic metamodel matching. The metamodel matching is obtained by means of the transformation of the participant metamodels into directed labeled graphs. The generated labeled graphs are compared by using a similarity algorithm to identify the equivalences between the different meta-elements, which correspond to the resultant metamodels matching. The obtained matching is used to transform models based on the source metamodel into equivalent models based on the target metamodel. This transformation can be performed by means of model transformation languages such as QVT. However, these kinds of approaches based on the direct transformation of source models into target models do not consider those elements of the source metamodel that have no representation in the target metamodel. Therefore, there is a high probability of lost modeling information in the transformation process. In addition, the precision of the proposed similarity identification algorithm is negatively affected by larger size metamodels, such as the UML metamodel that is considered in our example.

If we consider that one of the main objectives of the proposed approach is the interoperability of proprietary DSML and more general modeling languages, it is very common to find modeling elements in the proprietary DSMLs that are not present in more general modeling languages that are not centered on model compilation tasks. This is produced by the nature of proprietary DSMLs and generic modeling languages.

A modeling language that covers a domain in generic way, such as *i** for organizational analysis, provides a general vision of the constructs that are necessary for representing the target domain. However, at difference of proprietary DSML related to MDD approaches, these modeling languages are not centered on those details that are necessary to perform an appropriate model compilation process. This situation is clearly observed in general-purpose modeling languages, which are defined for their application into different domains. By contrast, a proprietary DSML related to an MDD approach is oriented to a specific domain, and it introduces particular modeling elements for a precise representation of the required software products in the domain context.

Thus, the compilation of the models defined with these DSMLs can be performed in an automatic way.

This gap between specific DSMLs for MDD approaches and generic modeling languages produces the loss of the specific MDD details when the DSML models are transformed. With the proposal presented in this chapter, we provide a solution to face the loss of modeling information during the transformation of DSML models. Thus, it is possible to obtain a complete integration of particular DSMLs and existing modeling languages through the bidirectional interchange of models. This solution is centered into the automatic generation of a UML profile that introduces into the more general modeling language those specific elements of the DSML that are lost in a direct model transformation process.

Proposals such as [130] state that it is possible to automate the interchange of models extended with a UML profile and DSML models by means of model transformations. However, this proposal requires the manual definition of a mapping between the participant metamodels. Since this mapping is defined independently of the definition of the UML profile, there exists the risk that the obtained mapping does not reflect the equivalences between the participant metamodels in a correct way. In our proposal this risk is avoided because the mapping is obtained during the UML profile generation. In addition, the complexity related to the correct design of a UML profile is encapsulated in the transformation of the Integration Metamodel [135]. Therefore, the mapping obtained from this transformation provides a correct identification of the equivalences between the generated UML profile and the DSML metamodel (using the Integration Metamodel as intermediate model).

7.5. Conclusions

In this chapter, an interchange proposal to obtain a hybrid modeling schema for the interoperation of modeling languages is presented. Thus, if we consider that the main resource in MDD technologies are the models, then, by applying the proposed interchange approach, it will be possible to achieve the interoperability in MDD processes. In addition, the tools related to existing modeling languages can be reused in the application of specific MDD solutions, thereby reducing the effort of implementing specific MDD tools.

For the application of this interchange approach, the previous steps of the proposed MDD interoperability (Integration Metamodel definition and UML Profile Generation) process play a fundamental role. In these steps, the artifacts and mapping information to automatically generate the interchange mechanisms are obtained.

This chapter also shows the application of the interchange proposal to a specific MDD approach, presenting an implementation schema that has been used to obtain the interoperability of UML in the OO-Method development process. In particular, the industrial implementation of OO-method, which is called *Olivanova The Programming Machine*. Thus, this implementation schema shows that it is possible to combine the use of generic and specific modeling tools for defining the conceptual models that are required by specific MDD approaches. But also, the application of specific MDD tools based on the proprietary DSML metamodels to automatically obtain different software products from models defined with a customized modeling language. In particular, the presented example briefly shows the application of UML in the OO-Method MDD development process, and how these customized UML models can be automatically compiled by using the Olivanova technology.

Some of the benefits for OO-method and UML that can be achieved with the presented implementation schema are:

1. The commercial structure defined for the OO-Method approach (such as the cost estimation based on functional size measurement) can be used by UML users.
2. The different tools based on the specific OO-Method DSML can be used in transparent way over extended UML models. This can be observed in the OO-Method model compiler [62], and functional size measurement tools [137, 138].
3. The users of the generated UML profile can easily migrate from UML tools to OO-Method tools. In this way, UML users that have already adopted the OO-Method approach can take advantage of the improved functionalities that the specific OO-Method modeling tools provide.

These benefits can be generalized to other MDD approaches that apply the proposed approach to interoperate with standard modeling languages and their supporting tools.

In the next chapters, the application of the proposed interoperability approach is detailed over two scenarios. These interoperability scenarios are the following:

1. The interoperability of UML in the OO-Method development process
2. The interoperability of *i** in the OO-method development process.

It is important to remark that the development of these two interoperability scenarios were relevant to improve and obtain the final interoperability approach presented in this thesis.

Chapter VIII.
Linking UML and MDD Approaches

Even though OMG in the Model-Driven Architecture (MDA) specification recommends the use of UML for model-driven developments, the lack of semantic precision in UML has led to different model-driven approaches proposing their own domain-specific modeling languages in order to introduce their modeling needs.

This chapter presents an interoperability scenario that is focused on the application of UML models into MDD processes. In particular, we face the customization of the UML association in order to facilitate its application in the OO-Method development process. To do this, the interoperability approach proposed in this thesis is applied. At the end of this chapter, we briefly present how the results obtained from the development of this interoperability scenario are used to generate software products through the industrial model compiler related to the OO-Method approach.

8.1. Introduction

The *Model Driven Development* (MDD) approach has achieved great relevance in the software industry, improving the software development process and reducing the cost of the developed applications [2]. In this context, one of the most widely used approaches is the *Model Driven Architecture* (MDA) [10, 11, 141], defined by OMG [34]. The MDA approach recommends the use of UML to define the conceptual models involved in MDD processes. However, UML is defined as a general purpose language with a flexible semantics that does not provide enough precision to define models that can be automatically transformed into complete software representations.

As states Milicev in [142], the association is one of the key constructs in UML for which a fully unambiguous semantics still does not exist. In early versions of UML, many authors have reported this issue, for instance, Graham et al. in [143] and Snoeck et al. in [144]. In the most recent versions of UML (UML 2.0 and above), this semantics has been somewhat improved, but some precision problems still persist. This situation is clearly reported by Albert et al. in [145] and Gueheneuc et al. in [146]. For instance, the behavior related to creation, deletion, or update of association instances, or a complete semantics for the *aggregation* relationships are not clearly specified [31].

In order to provide an effective solution for interoperating UML and MDD processes, this chapter shows the application of the MDD interoperability approach proposed in this thesis. This application allows the UML syntax (proposed in the UML specification) to be adapted to the modeling needs of a specific MDD approach. In particular, we advocate on showing how to extend (customize) the abstract syntax of the UML constructs that are related to specifying association relationships among classes.

For the application of the interoperability approach, we have inherited the modeling aspects related the *OO-Method* approach [16]. We consider that this is a suitable option for demonstrating the effectiveness of our proposal since OO-Method is an object-oriented MDD method that has been successfully applied to the software industry[3].

[3] The industrial implementation of OO-Method has been successfully applied in several companies such as Toshiba, Daimler-Chrysler, and Repsol.

Thus this chapter makes a twofold contribution:

1. It shows how a correct integration of the syntax that supports the proposed semantics can be performed by the application of the proposed interoperability process
2. It presents an industrially-tested semantics that can be used as a reference for the application of the UML association in MDD environments

The chapter also exemplifies how to obtain a final software product from a UML model that has been extended with the generated UML profile. This model compilation is performed using the industrial solution that implements the OO-Method approach [63] and the interchange tools that were implemented by applying the interchange approach proposed in Chapter VII.

The rest of this chapter is organized as follows: Section 8.2 presents a background of the concepts and technologies involved. Section 8.3 introduces the semantics adopted in this interoperability scenario to improve the UML association for MDD processes. Section 8.4 shows how the customization of the UML association is performed. Section 8.5 presents a model compilation example related to a UML model that has been extended with the proposed semantics. Finally, Section 8.6 presents our conclusions.

8.2. The applicability of the UML association in MDD Processes

This section is centered on the need of customizing the UML specification for its appropriate application in MDD processes. Specifically, we show why the UML association must be adapted for this purpose.

In general terms, UML specifications include association definitions that do not achieve a consensus for a unified semantic definition. Several works have appeared highlighting the drawbacks of the language and trying to answer many important questions concerning associations. For instance, the works presented by Diskin and Dingel [147], Genova et al. [148], and Milicév [142]. With regard to the UML1.4 specification [149], Henderson-Sellers et al. have presented different works [150-152] searching for answers to some relevant questions, such as the directionality of associations or the special meaning of aggregation and composition. Special attention to the whole-part properties of the

association has been given by Barbier and Belloir et al. in [153] and [154]. Also, in [155], Stevens tries to clarify some confusing concepts regarding associations (without using formalizations), such as the use of *tuple* for defining links, some complex questions of the multiplicity definition, or the static and dynamic notion of associations. This last concept is also discussed by Genova et al. in [148], where the authors propose a new classification for associations. With regard to most recent versions of the UML specification (UML 2.x), it is recognized that the association definition has been improved, but some problems still persist [145, 146]. For instance, Diskin in [147] presents a framework to formally explain several confusing notions of associations and detects some flaws in the association part of the UML metamodel. In [143], Graham et al. center on newer concepts that are related to association ends in order to improve the expressiveness of the UML association. In addition, such as France et al. clearly state in [31], there exists a well-known gap between the conceptual representation of the UML association and its correct implementation in final software products, which is a relevant issue for the correct application of UML in MDD processes. This situation is also present in some implementation proposals for the UML association such as the proposed by Akehurst et al. [156] and Gessenharter et al. [157], where elements represented at the implementation level have no correspondence at the conceptual level. In our proposal, we have centered our attention on a MDD approach called OO-Method.

The OO-Method approach puts into practice most of the ideas presented in the analyzed works [14]. However, unlike most of the works that just focus on specific parts of the association definition, OO-Method provides a holistic view of the association. For instance, some works just face the composition definition, which is a subtype of the association; others focus on the notation for specifying associations; and others on the alternatives to implement associations. Instead, OO-Method integrates all these aspects to obtain a complete association specification. Moreover, even though some of the analyzed proposals have a certain level of technology support, they are mostly applied at the theoretical and academic levels. In contrast, the OO-Method approach has been successfully applied to the software industry, which demonstrates the effectiveness of this approach to support real software development projects. Thus, the rigorous semantics of the OO-Method association encourages the use of this approach to explain how this relevant UML construct can be customized for effective MDD application.

In the OO-method approach, the class model is the core of the OO-Method conceptual model; the rest of the models involved are defined starting from elements of the class model. The constructs involved in the specification of associations among classes are defined within the OO-Method class model. Moreover, the correct specification of the different modeling aspects of the association is probably one of the most important and complex parts of the OO-Method class model. For this reason, the OO-Method association has been chosen to apply the interoperability approach presented in this thesis.

Thus, by integrating the modeling aspects of the OO-Method association into UML, we obtain an extended UML association that provides appropriate modeling information for its application in the OO-Method MDD process. To perform this integration according to the MDA guide [10] and the latest UML specification [35, 122], the definition of a UML profile is the more suitable option. In this context, the application of our MDD interoperability approach that is based on the UML profile generation is a recommended alternative to achieve the UML and OO-Method interoperability.

8.3. The Semantics Proposed to Customize the UML Association

This section introduces the semantics proposed to customize the UML association, which is inherited from the OO-Method approach. However, since many concepts in OO-Method already exist in the UML specification, we only focus on the aspects that meaningfully contribute to improving the UML association in the context of the MDD development process. Marin, et al. in [158] show a detailed case study of the OO-method approach, where the modeling flexibility of the OO-Method association semantics can be observed.

In the OO-Method context, an association is defined as a structural relationship between classes that represents connections (*links*) between the objects of these classes (*participant classes*). OO-Method associations are binary, so they only have one or two participant classes (one class in the recursive associations). Thus, the association concept used in this chapter always refers to binary associations.

The *association ends* are the endpoints of an association, which connect the association to its participant classes. The name of an association end corresponds

to the *role* of that end (the task that the participant class plays in the association). The association ends are characterized by the multiplicity property, which specifies the maximum/minimum number of objects that can/must be connected to an object of the opposite end. The relevant concepts that must be added to UML for appropriate definition of associations according to the OO-Method approach are:

- Unique identification for class instances (objects).
- Precise behavior for aggregation and composition concepts.
- Precise behavior related to creation, deletion and update of links.

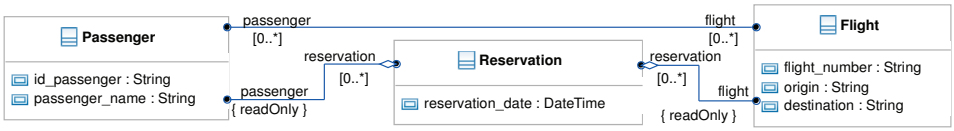

Figure 49. Example UML model

Figure 49 shows a brief UML model that is used throughout this chapter to illustrate our proposal. This model was defined using the *Eclipse UML2* tool [126]. It shows an association between the classes *Passenger* and *Flight*, and an aggregation between these two classes and the class *Reservation*. A passenger can make a reservation for a specific flight, or can take a flight without a previous reservation. The association between the classes *Passenger* and *Flight* indicates those passengers that actually flew. Thus, a passenger with a reservation may not be related to a flight, for instance, if the passenger misses the flight.

Following, the specific modeling features that are integrated into UML are presented.

8.3.1. Object Identification

In UML, it is not possible to uniquely identify the objects participating in an association when the model is instantiated, since the UML specification does not define a mechanism for the identification of class instances. It is interesting to observe that, even though the correct identification of objects is a relevant issue for correcting compilation of the association, proposals that deal with the compilation of the UML association usually omit this feature [156, 157]. To solve this problem, the concept of *Identification Function* is introduced in the OO-Method approach.

The Identification Function corresponds to a set of structural properties that allows the unique identification of class instances. Thus, the Identification Function is specified by means of a set of attributes (one or more) owned by a class. In the example shown in Figure 49, the attribute *id_passenger* is a clear candidate to conform the Identification Function of the class *Passenger*; the same is true for the attribute *flight_number* of the class *Flight*. The Identification Function is specified by adding the Boolean property *isIdentifier* to the specification of class attributes. Thus, *isIdentifier* is set to *TRUE* when a class attribute participates in the Identification Function of the owning class.

Figure 50 shows how the Identification Function is used for the identification of the objects participating in an association, and how this feature allows the unequivocally identification of links.

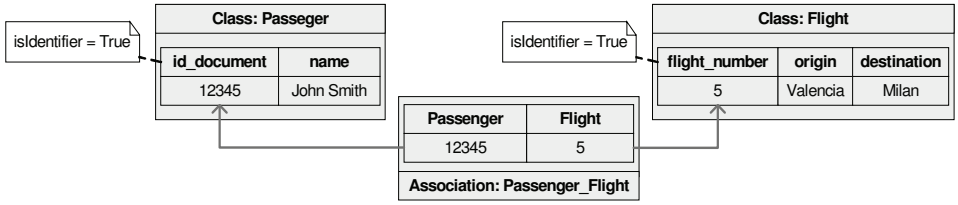

Figure 50. Example of Identification Function in the association

8.3.2. Aggregation and Composition

In order to make the semantics of the aggregation concept more precise, we adopt the following UML assertion: *"An association may represent an aggregation (i.e., a whole/part relationship). In this case, the association-end attached to the whole element is designated, and the other association-end of the association represents the parts of the aggregation. Only binary associations may be aggregations"*.

In OO-Method, this definition is extended with additional semantics. Thus, the property of *Identification Dependency* of the whole with regard to the part is introduced to represent the dependency that exists between composite (whole) and component (part) classes. This dependency is discussed in works such as the presented by Barbier et al. in [153].

The identification dependency implies that the identifier of the composite class is built using the identifier of the component classes (and, if necessary by adding some attribute of the composite class). According to UML, the lower cardinality of the association end related to the component class must be 1, that

is, [1..x] cardinality. However, to guarantee the correct compilation of the identification dependency in the OO-Method development process, the upper cardinality of the association end related to the component classes is constrained to 1, that is, [1..1] cardinality. This constraint is defined since [1..*] cardinality implies that the identification function of the composite class must be conformed by a multi-valued attribute. However, the target implementation platforms of the OO-Method approach are based on relational databases that do not provide support for multi-valued attributes, such as SQL Server.

In the aggregation example presented in Figure 49, the component classes are *Flight* and *Passenger* (with cardinality [1..1]) and the composite class is *Reservation* (with cardinality [0..*]). The Identification Function of *Reservation* is composed by the attribute *id_passenger* from *Passenger* and *flight_number* from *Flight*[4].

An aggregation can be specialized in a *composition (composite aggregation* in UML*)*, which presents additional features. These features are the following:

- A part must be included in, at most, one composite at a time.
- If a composite is deleted/modified, all its parts are deleted/modified with it.
- There is an identification dependency of the part with regard to the whole. Thus, for the composition, the cardinality related to the composite class must be [1..1]. Note that the identification dependency of composition is opposite to the identification dependency of the aggregation.

8.3.3. Creation, Deletion, and Modification of Links

The creation, deletion, and modification of links are only required in *dynamic* associations. In OO-Method, an association is dynamic when the two participant association ends are defined as dynamic. Otherwise, the association is not dynamic since the links established cannot change throughout the whole life of the participant objects. Figure 51 exemplifies this situation for a non-dynamic association.

Figure 51 shows an association between the classes *A* and *B*. The association end related to class *A* is *static*, and the association end related to class *B* is

[4] The specification of the identification function for the class *Reservation* is not required, since it can be automatically inferred from the aggregated classes during the model compilation process.

dynamic. In *T1*, a new instance of class *B* (named *b1*) is created. The instance *b1* is linked with the two existing instances *a1* and *a2* of class *A*. In *T2*, the instance *b2* is created and is also associated with *a1* and *a2* (changing the set of links that are defined for *a1* and *a2*). These links can be performed because the association end *RoleB* is *dynamic*. In *T3*, a new instance of class *A* (named *a3*) is created, and it is associated with *b1* and *b2*. However, a3 cannot be linked to these two instances of class B because the association end *RoleA* is static, and the links defined in the creation of the instances *b1* and *b2* cannot change during their lives. Furthermore, this implies that the links established between the instances of classes *A* and *B* during the creation of the instance *b1* and *b2* cannot be changed, and they can only be deleted when one of the participant instances is deleted.

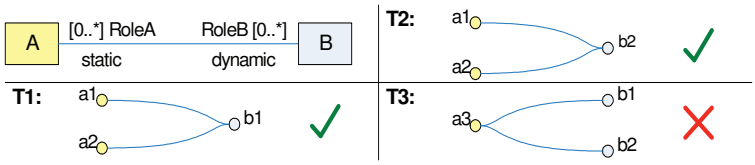

Figure 51. Example of association temporality

In a UML model, a dynamic association can be represented as a binary association where the property *readOnly* of the two involved association ends are set as *false*. Thus, a dynamic association implies that the links established between instances related to the two participant ends can be changed (inserted, deleted, or updated) during the life of the participant objects.

In UML, it is necessary to manually define (in the participant classes) operations to represent the management (creation, deletion, and modification) of links. These operations must include the specification of the related behavior, which can be specified using different languages, such as natural language, *Action Semantics* [159], or the *Object Constraint Language* (OCL) [37]. For the definition of these operations, it must be taken into account that the management of links simultaneously affects properties of the two participant classes of the association. Therefore, the involved operations must be simultaneously executed in the participant classes. It is possible to observe that the definition of these operations is not trivial, and, hence, the manual definition of the behavior that is related to these operations makes the correct specification of the associations difficult and error-prone, which is of great relevance when this specification is interpreted by an automatic model compiler. To face this issue, certain works [156, 157] have proposed a direct implementation of operations related to

controlling the associations' behavior. In these proposals, the operations related to the management of links are represented at the implementation level (programming code). Nevertheless, since these operations do not have representation at the conceptual level, customization of the behavior related to links management cannot be performed in the UML model.

Thus, our proposal introduces the concept of *shared event* to represent these linking management operations. A shared event is a special kind of operation that defines the behavior related to dynamic associations. It is owned by the two participant classes, and its definition can be separately customized in each participant class. Thus, a shared event has a definition that is distributed between the classes that participate in the association. Events of this kind always require two input parameters, either linked objects or objects to be linked. A shared event can be of three types:

- **Insert Event:** creates a link between an object at one end of the association and an object at the opposite end.
- **Delete Event:** removes an existing link between an object at one end of the association and a related object at the opposite end.
- **Edit Event:** changes an existing link between an object at one end of the association and a related object at the opposite end.

The types of shared events depend on the cardinality of the association ends. Cardinality [1..1] in an association end prevents an existing link from being deleted or a new link from being created, that is, the execution of an insert or delete event is not possible. In this case, a link is established during the creation of an object at the opposite end, and it can only be deleted when one of the participant objects is deleted. Hence, the dynamic association can only be managed by an *edit shared event* because the only option is to change the existing link for another one. Thus, the edit shared event has the effect of a simultaneous execution of an insert and deletion event, which prevents the violation of the association cardinality. In any other case, the dynamic associations are managed by the *insert* and *delete* events.

The behavior of a shared event can be customized by defining preconditions and post-conditions, which must be specified by means of well-formed, first-order logic formulas. The case study presented in [158] provides more detailed examples about the customization and integration of shared events in more complex services.

8.4. Integration of the Proposed Semantics into UML

In this section, we integrate the semantics proposed (in the previous section) into UML by applying the first three steps of the MDD interoperability process (see Chapter IV). By means of this *Integration Process,* we obtain the following outputs:

1. A specific metamodel that defines the abstract syntax required for representing the proposed association semantics.
2. The UML profile that integrates our proposal into UML.
3. The set of mappings to perform the automatic generation of model interchange mechanisms.

The metamodel that represents the required abstract syntax is defined as an *Integration Metamodel* [127] according to the second step of the integration process (see Chapter V). In this case, we have directly defined an Integration Metamodel for the proposed semantics, since there is no a previous EMOF metamodel for the OO-Method industrial approach. The implemented OO-method tools are based in a XML definition that is not based on a standard metamodel specification. In other words, the metamodel related to the industrial OO-Method approach is implicit in the OO-method compilation process, but is not explicit in a formal specification that could be computationally interpreted, such as an *Ecore* file [57]. This is one of the reasons of why the interchange tools for the industrial OO-Method technology are implemented by means of XSLT transformations.

The Integration Metamodel defined for the proposed semantics allows the related abstract syntax to be integrated into UML by means of a UML profile that is automatically generated [135]. This generation is performed according to the third step of the proposed interoperability approach (see Chapter 5), which is oriented to obtain the corresponding UML Profile from the Integration Metamodel.

Finally, the interchange mechanisms are automatically generated from the mappings obtained during the generation of the UML profile.

8.4.1. Integration Metamodel definition

Since the industrial OO-Method approach has not a standard metamodel definition, we have directly defined the Integration Metamodel to support the

abstract syntax related to the proposed OO-Method semantics. In other words, we have applied the first and second step of the interoperability approach at the same time.

According to the Integration Metamodel specification (see Chapter V), there are four conditions that an Integration Metamodel must hold for the automatic generation of the metamodel extensions. These are the following:

- All the classes from the Integration Metamodel are mapped to class of the UML Metamodel. This assures that the constructs from the MDD approach can be represented from the UML constructs.
- The mapping is defined between elements of the same type (classes with classes, attributes with attributes, and so on).
- An element from the Integration Metamodel is only mapped to one element of the UML Metamodel.
- If the properties of a class A from the Integration Metamodel are mapped to properties of a class B of the UML metamodel, then the class A is mapped to the class B or a specialization of it.

Figure 52 shows the Integration Metamodel that describes the abstract syntax for the semantics introduced in section 8.3. This corresponds to a subset of the whole metamodel that is necessary to describe the abstract syntax of the industrial implementation of the OO-Method MDD approach.

The Integration Metamodel presented has been specified using the Eclipse UML2 Tool [126] since it provides automatic generation of EMF metamodels from the defined UML2 metamodels. EMF [160] is the *Eclipse Modeling Framework*, which is based on the EMOF specification. Also, the generated EMF metamodels are tagged with additional information to automatically obtain model editors that have interpreters for the defined OCL rules and that support UML profile extensions. Additionally, the Eclipse UML2 project provides a complete implementation of the UML metamodel, which is defined according to the official UML specification. This facilitates that the artifacts involved in the application of our proposal fulfill the OMG standards.

According to the defined Integration Metamodel (Figure 52), the classes can own *data-valued* attributes (class *DataValuedAttribute*) or *object-valued* attributes (class *ObjectValuedAttributes*). The data-valued attributes are the typical class attributes, such as the name of the passenger in the UML example (Figure 49). In the class *DataValuedAttribute*, the attribute *isIdentifier* indicates whether the

data-valued attribute participates as (part of) the identifier of its owning class (*isIdentifier* = True), and the attribute *nullsAllowed* specifies whether the data-valued attribute can take null values.

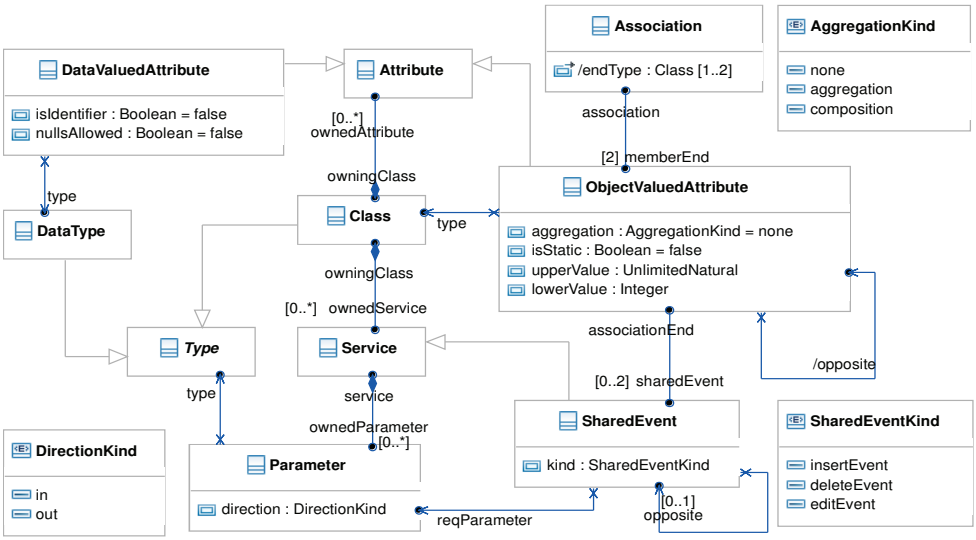

Figure 52. : The Integration Metamodel of the proposed association semantics

A data-valued attribute that participates as an identifier of a class cannot take null values (*nullsAllowed* = False). An object-valued attribute represents an association end, such as the *passenger* and *flight* related to a *reservation* in the UML example of Figure 49, and it is related to the association by means of the association *memberEnd*. In the context of binary associations, the object-valued attributes must always have an opposite association end (association *opposite*). In the class *ObjectValuedAttributes*, the attribute *aggregation* indicates (if necessary) the kind of association (*aggregation*, *composition*, or *none*) related to the association end. The class related to an association end is indicated by the association *type*. In an object-valued attribute the attribute *isStatic* indicates if the association end is static (*isStatic* = True) or dynamic (*isStatic* = False). The name of the object-valued attribute (which is inherited from the class *NamedElement)* indicates the *role name* of the corresponding association end. The class *NamedElement* and the related inheritance hierarchy are not represented to simplify the Integration Metamodel diagram since all the classes defined are specializations of *NamedElement*.

Table 5 shows the OCL rules that object-valued attributes must fulfill to ensure that the cardinality constraint related to the Identification Dependency feature of aggregation and composition is not violated.

Table 5. OCL constraints for association ends

Description	If the attribute *aggregation* of an association end holds the value *#aggregation*, then the opposite end must have cardinality [1..1].
Context	ObjectValuedAttribute
OCL	self.aggregation = #aggregation implies self.opposite.lowerValue = 1 and self.opposite.upperValue = 1
Description	If the attribute *aggregation* of an association end holds the value *#composition*, this end must have cardinality [1..1].
Context	ObjectValuedAttribute
OCL	self.aggregation = #composition implies self.lowerValue = 1 and self.upperValue = 1

The definition of a shared event is distributed between the classes that participate in the association because a shared event simultaneously changes properties of the participant objects. To support this semantics, in the Integration Metamodel a shared event is represented as two dependent events, which are related by means of the association *opposite*. Table 6 shows the OCL constraints defined for shared events.

A shared event must always participate in an association and is only related to one object-valued attribute. This is represented by the association *associationEnd* with the cardinality [1..1].

The association *ownedParameter* (inherited from *Service*) represents the parameters of a shared event. One of the two parameters that are required by a shared event (the two participant objects) is the object that executes the service, whose type corresponds to the class that owns the service. However, this parameter does not need to be defined in the model because it is implicit in the semantics of the service and it can be inferred from the association *owningClass*. The second parameter required by a shared event (the participant object of the opposite association end) is identified by the association *reqParameter*.

According to the Integration Process, to complete the definition of the Integration Metamodel it is necessary to identify the equivalences between this metamodel and the UML metamodel. 0 presents these equivalences.

Table 6. : OCL constraints for shared events

Description	Shared events can only be defined when both association ends are dynamic.
Context	ObjectValuedAttribute
OCL	self.temporality = #static or self.opposite.temporality = #static implies self.sharedEvent->isEmpty()
Description	Only the shared event *editEvent* can be defined for dynamic associations when one of the association ends has cardinality [1..1].
Context	ObjectValuedAttribute
OCL	((self.temporality = #dynamic) and (self.opposite.temporality = #dynamic) and (self.lowerValue = 1) and (self.upperValue = 1)) implies self.sharedEvent.kind = #editEvent and self.sharedEvent->size() = 1
Description	The *insert* and *delete* shared events must be defined for dynamic associations when both association ends have cardinality [x..*].
Context	ObjectValuedAttribute
OCL	((self.temporality = #dynamic) and (self.opposite.temporality = #dynamic) and (self.upperValue > 1) and (self.opposite.upperValue > 1)) implies self.sharedEvent->size() = 2 and self.sharedEvent->exists(se | se.kind = #insertEvent) and self.sharedEvent->exists(se | se.kind = #deleteEvent)
Description	A shared event requires an opposite shared event with the same name and kind, except in the case of recursive associations.
Context	SharedEvent
OCL	self.associationEnd.type <> self.associationEnd.opposite.type implies self.opposite->notEmpty() and self.kind = sel.opposite.kind and self.name = self.opposite.name
Description	In recursive associations, a shared event does not have an opposite event.
Context	SharedEvent
OCL	(self.associationEnd.type = self.associationEnd.opposite.type) implies self.opposite->isEmpty()
Description	A shared event cannot be opposite to itself.
Context	SharedEvent
OCL	self.opposite->notEmpty() implies self <> self.opposite

Table 7. Equivalences between the Integration Metamodel and the UML Metamodel

Integration Metamodel	UML Metamodel	Integration Metamodel	UML Metamodel
AggregationKind	AggregationKind	ObjectValuedAttribute	Property
.none	.none	.type	.type
.aggregation	.shared	.association	.association
.composition	.composite	.opposite	.opposite
Association	Association	.aggregation	.aggregation
.memberEnd	.memberEnd	.isStatic	.isReadOnly
Attribute	Property	.upperValue	.upper
.owningClass	.class	.lowerValue	.lower
Class	Class	Parameter	Parameter
.ownedAttribute	.ownedAttribute	.direction	.direction
.ownedService	.ownedOperation	.service	.operation
DataType	DataType	.type	.type
DataValuedAttribute	Property	Service	Operation
.type	.type	.owningClass	.class
DirectionKind	ParameterDirectionKind	.ownedParameter	.ownedParameter
.in	.in	SharedEvent	Operation
.out	.out	Type	Type

8.4.2. UML Profile Generation

The third step of the Integration Process involves the generation of the UML profile. It requires a comparison between the Integration Metamodel and the target Metamodel (UML metamodel in the example) to identify the structural differences that exist between these two metamodels. These differences correspond to the metamodel extensions that must be implemented in the UML profile generation. With regard to the conditions established for the Integration Metamodel definition and the mapping information presented in 0, this second step of the process is automatically performed. This prevents the extra time, effort, and potential errors that are involved in a manual identification of the required extensions. Table 8 summarizes the results of this comparison for the presented example.

Table 8. Comparison between the Integration Metamodel and the UML Metamodel

Integration Metamodel	Difference
Association	
.memberEnd	lower bound (IM = 2; UML = *)
DataValuedAttribute	
.type	type (IM = DataType; UML = Type)
.isIdentifier	*new*
.nullsAllowed	*new*
ObjectValuedAttribute	
.type	type (IM = Class; UML = Type)
.association	lower bound (IM = 1; UML = 0)
.opposite	lower bound (IM = 1; UML = 0)
.sharedEvent	*new*
SharedEvent	
.kind	*new*
.opposite	*new*
.reqParameter	*new*
.associationEnd	*new*
SharedEventKind	*new*

In Table 8, the column *Integration Metamodel* shows the *elements* of the Integration Metamodel that differ from the UML metamodel elements, and the column *Difference* shows what the differences are by indicating the values that differ for the Integration Metamodel element (*IM*) and the UML metamodel element (*UML*). The word *new* in the column *Difference* indicates when the Integration Metamodel introduces an element that does not exist in the UML metamodel. Thus, the elements that must be introduced into UML to solve the identified differences are the extensions that must be defined in the UML profile.

After the metamodel comparison has been performed, the final UML profile is automatically generated (see Figure 53). The UML profile generation is performed by means of the set of transformation rules presented in Chapter VI, which are applied over the Integration Metamodel defined. These rules are applied taking into account the equivalences presented in 0 and the differences presented in Table 8.

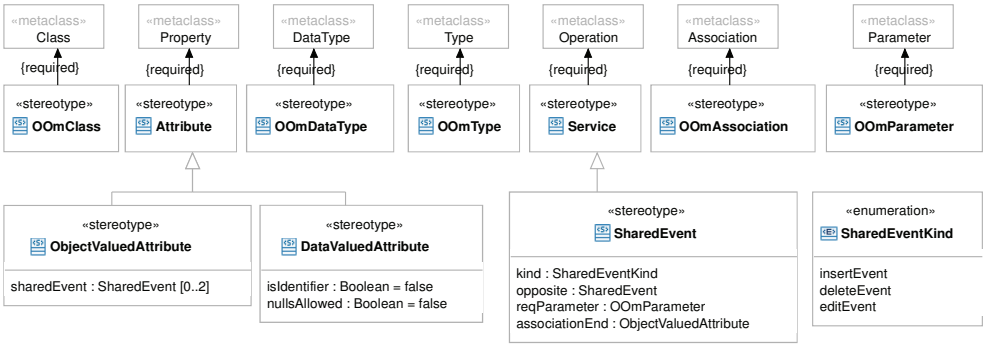

Figure 53. UML Profile generated from the defined Integration Metamodel

The main features of the transformation rules that are related to the UML profile generation are the following:

- Equivalent classes of the Integration Metamodel are transformed into stereotypes that extend the corresponding UML class identified in 0. If the class of the Integration Metamodel has the same name as the corresponding UML class, then a prefix is added to differentiate the name of the stereotype from the name of the extended class. In the example, the prefix *OOm* is used. For instance, the class *Association* of the Integration Metamodel generates the stereotype *OOmAssociation* (prefix + class name).

- The *new* properties (attributes and associations) that are identified in the metamodel comparison (see Table 8) are represented as tagged values. For instance, the attribute *isIdentifier* of the class *DataValuedAttribute* is defined as a tagged value in the stereotype *DataValuedAttribute* (see Figure 53).

- Differences between equivalent properties are managed with OCL constraints. For instance, the lower bound difference of the associations *opposite* and *association* of the class *ObjectValuedAttribute* (see Table 8) are managed with an OCL rule with the following structure:

  ```
  self.[property]->size() >= [newLowerBound]
  ```

  The OCL rules are defined in the stereotypes that are generated from the involved classes; in this case, the stereotype *ObjectValuedAttribute*.

- The generated stereotypes have all the constraints defined in the transformed classes. For each constraint, the elements of the Integration Metamodel are replaced by the corresponding elements of the UML metamodel or by

elements of the generated UML profile in the case of new elements. For instance, the first OCL rule defined in Table 5 is defined in the stereotype *ObjectValuedAttribute* as follows:

```
self.aggregation = #shared implies
self.opposite.lower = 1 and self.opposite.upper = 1
```

In the presented OCL rules, the elements written in *italics* indicate the UML elements that are used to replace the corresponding Integration Metamodel elements according to the equivalences presented in 0.

8.5. Compiling the Extended UML Association

Figure 54 shows the example UML model extended with the generated UML profile, where the application of the different stereotypes can be observed. The services for the management of links between objects are defined as *shared events* by means of the stereotype *sharedEvent*, the service *new_association* is defined as an *insert event*; and the service *del_association* is defined as a *delete event.*

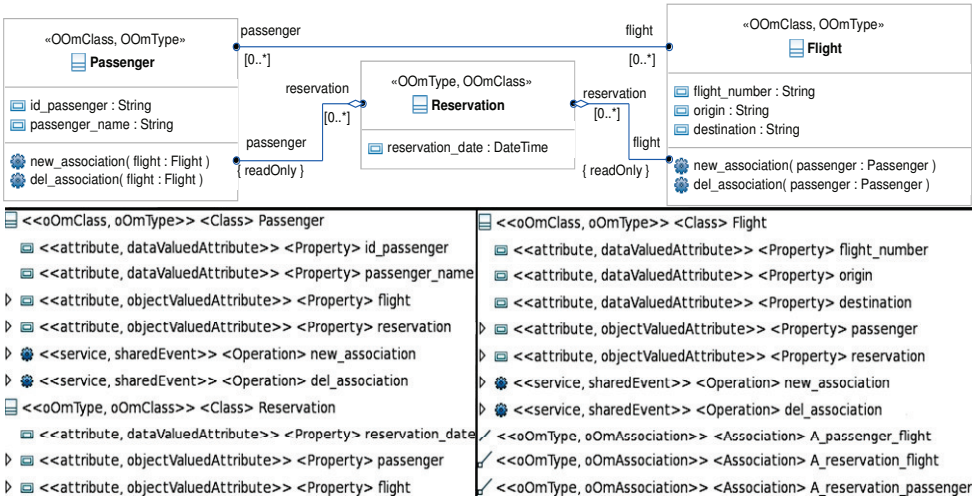

Figure 54. Example UML model extended with the generated UML profile

Once the UML model is correctly specified, it is compiled with the OO-Method model compilation technology [62] by using the specific interchange

proposal presented in Chapter 6 [18, 132], which is based on the generated UML profile and the mapping information obtained during the application of the interoperability process. In the compilation of the UML model, default services for the creation, deletion, and edition of instances are automatically created for each class. These default services are not created for those classes that already have these kinds of services defined in the UML model.

It is important to remark that the extensions introduced in the UML model provide a precise definition for the association at conceptual level, which allows the independency of the business-logic layer from the implementation platform. With regard to this independence, the OO-Method compilation technology can generate applications for different implementation platforms from the same UML model. For instance, from an OO-Method model, the industrial OO-Method implementation can currently generate software products in *.Net*, and *J2EE* developing platforms, and *Oracle*, *SQL Server*, *DB2*, and *PostgreSQL* database servers. The case study presented in [158] shows how the target platform is selected by using a specific configuration tool for the model compilation process. Figure 55 shows an screenshot of the configuration tool for the compilation process, where the different compilation alternatives that the Olivanova technology provides for the imported UML model can be observed.

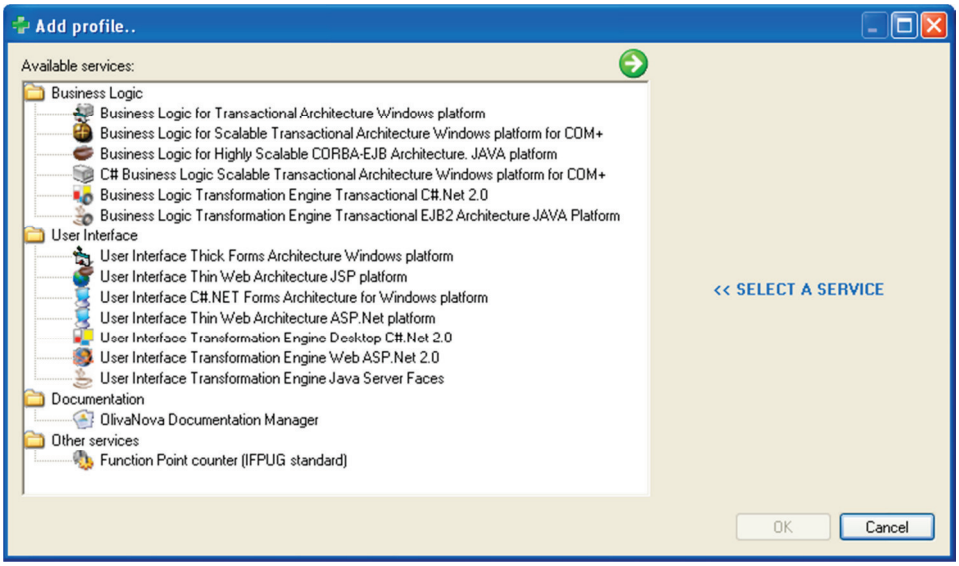

Figure 55. Compilation alternatives provided by the Olivanova technology

Next, the generated application is briefly explained considering the aspects related to the compilation of the object identification, the compilation of the aggregation, and the compilation of the shared events.

8.5.1. Compilation of the object identification

The Identification Function has a direct impact on the database generated in the compilation of the UML model (see Figure 56). In this database, the identification function of each class is transformed into a primary key of the table that corresponds to the compiled class. In addition, the table *TM_PassengerFlight* (not present in the UML model) has been automatically generated for the implementation of the many-to-many association that exists between the classes *Passenger* and *Flight*.

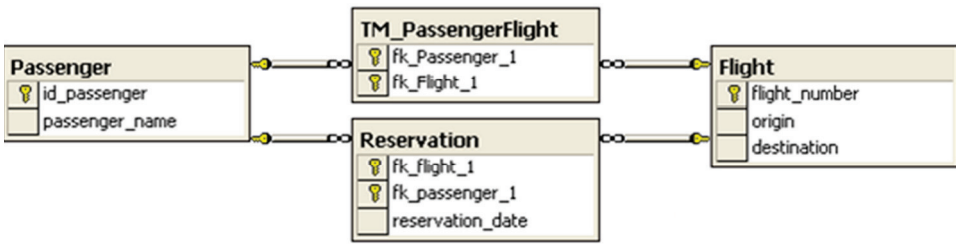

Figure 56. Diagram of the SQL database generated from the example UML model

8.5.2. Compilation of the Aggregation

In the example, the class *Reservation* is aggregated by *Flight* and *Passenger*. The aggregation implies the identification dependency of the whole with regard to the part. Figure 56 shows that to implement the identification dependency, the primary key of the table *Reservation* is comprised of the primary keys of the tables *Passenger* and *Flight*. This implementation is automatically inferred by the model compilation process from the defined aggregation relationships. In addition, the cardinality [1..1] in the component side is mandatory according to the semantics proposed for the identification dependency.

The cardinality [1..1], which is defined for the association ends related to the classes *Flight* and *Passenger*, implies that a reservation must always be related to a flight and a passenger. Thus, the links between a reservation and the corresponding passenger and flight must be created at the same time as the creation of the reservation. This particular semantics of the aggregation must be

considered in the compilation of the service that creates instances of the class *Reservation*. Figure 57 shows a screenshot of the application generated from the UML model, which is related to the execution of the service for creation of new reservations.

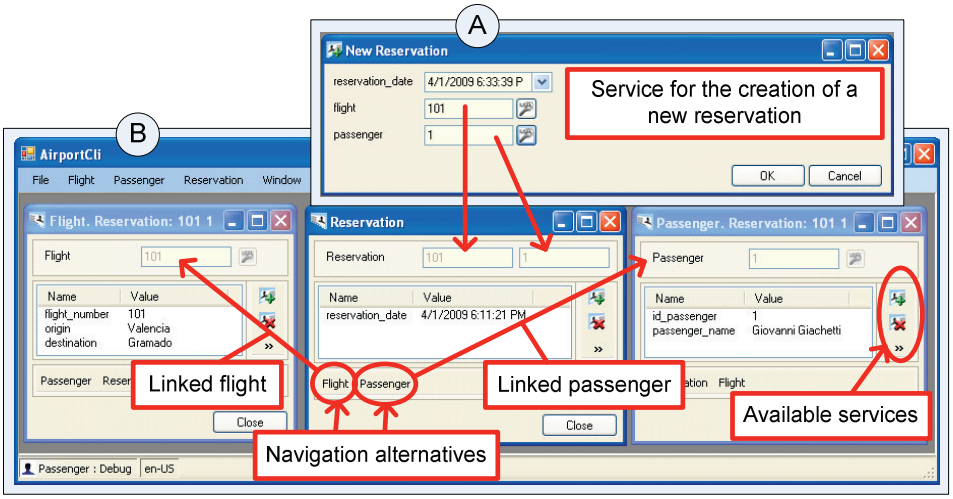

Figure 57. Execution of the service that creates new instances of the class Reservation

Figure 57-A shows the form related to the execution of the service, and Figure 57-B shows the result of the execution, which indicates the flight and the passenger linked during the creation of the reservation. The navigation alternatives have been inferred from the associations defined in the UML model.

Figure 57-A also shows that the service for the creation of an instance of the class *Reservation* has three inbound arguments: the date of the reservation (defined in the UML model) and the flight and passenger related to the new reservation (inferred from the aggregations relationships).

8.5.3. Compilation of shared events

Figure 58 shows a screenshot of the application related to the execution of the shared event *del_association* (executed from an instance of *Passenger)*. This shared event has two input arguments: the identifiers of the linked passenger and flight. The execution of the shared event *del_association* destroys a link that exists between the selected objects (the input arguments).

Specific behavior for the management of links is defined at the conceptual level by means of the customization of the defined shared events. Thus, in the shared event *del_association*, we define the precondition: *a link between a passenger and a flight cannot be destroyed if there already exists a reservation related to these two objects.*

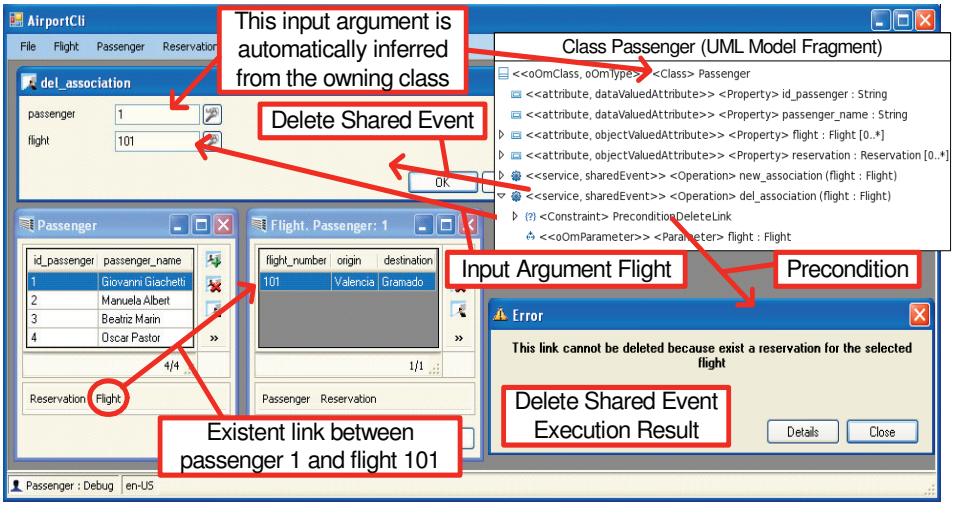

Figure 58. : Execution of the service *del_association* with a precondition

Figure 58 shows that the execution of the service *del_association* (from an instance of the class *Passenger*) does not fulfill the precondition because there already exists a reservation for the selected passenger and flight; therefore, an error-message form is displayed. This situation shows the relevance of the Identification Function, since the identifiers of the instances of the classes *Passenger* and *Flight* are used in the detection of pre-existent links.

8.6. Conclusions

This chapter has presented two main contributions. The first of these is the application of the proposed process to interoperate UML and MDD approaches in order to allow the automatic compilation of UML models by using existing MDD technologies. In particular, we advocate customizing the semantics of the UML association with a precise semantics that is obtained from the industrial

implementation of the OO-Method approach [12]. The abstract syntax that supports this semantics is defined using an Integration Metamodel [127], which allows the automatic generation of a UML profile that integrates the conceptual constructs and properties required for the proposed semantics into UML. This reduces the complexity related to the correct specification of UML extensions, providing an advantage over a manual UML profile specification, which is a time-consuming and error-prone task [17]. Furthermore, the proposed process takes advantage of the OMG standards and existing open-source tools such as [80, 126, 134], which facilitate the interchange of knowledge within the MDD community.

The second main contribution of this paper is the semantics proposed for the UML association and the UML extensions generated to support this semantics. These extensions provide the capability of precisely representing the structure and behavior of the association at the conceptual level, which can be used to automatically obtain a software product in different implementation platforms. This distinguishes our proposal from others that provide a direct implementation of the UML association in a specific language [156, 157], which require a specific behavior for the associations to be defined directly in the code. Thus, the proposed association modeling representation can be used as a reference by different MDD approaches.

The interoperability scenario that is presented in this chapter is based on the integration of two modeling approaches that are in the same abstraction level, but also, that share a very similar metamodeling structure. The two considered approaches are related to class model representations. Hence, this first scenario could seem trivial to perceive the real potential of the proposed integration approach.

Therefore, in order to stress the proposed integration approach, we have defined a second interoperability scenario, which is oriented to integrate two modeling approaches that are in a different abstraction levels and application domains. These modeling approaches are: 1) the *i** framework, which is related to the organizational analysis domain; and 2) the OO-Method approach, which is related to the software system modeling.

This new interoperability scenario is focused on show how the proposed integration approach can be applied in a more complex integration situation, which demonstrates the potential of the proposal in a non-trivial integration schema. The next chapter details the implementation of this second interoperability scenario and the results obtained.

Chapter IX.
Linking Goal-Oriented Modeling and Model-Driven Development

In the context of Goal-Oriented Requirement Engineering (GORE) there are interesting modeling approaches for the analysis of complex scenarios that are oriented to obtain and represent the relevant requirements for the development of software products. However, the way to use GORE models in automated Model-Driven Development (MDD) processes is not clear, and, in general terms, the translation of these models into the final software products is still manually performed.

This chapter presents a scenario where the MDD interoperability approach proposed in this thesis is used to automatically link GORE models and MDD processes. This scenario has been elaborated by considering the experience obtained from linking the i framework with the industrial implementation of the OO-method approach. Special attention is paid to the generation of appropriate model transformation mechanisms to automatically obtain an initial MDD model from a GORE model, and how to specify verification mechanisms to assure the correct transformation of the models involved.*

9.1. Introduction

An appropriate requirement specification is a key aspect for the correct development of software systems [161, 162]. Requirement specification should not only include software specifications, but also multiple complementary views: intentional, structural, responsibility, functional, and behavioral. The requirement engineering (RE) field offers different modeling approaches that analyze complex scenarios and elicit their relevant requirements [162-164]. Of these approaches, the Goal-Oriented Requirement Engineering (GORE) plays a significant role [161, 165] because it is mainly concerned with the stakeholder intentions and their rationales. However, the way in which GORE models should be used in an automated Model-Driven Development (MDD) process [1] is very often too vague. An important issue that is still pending is how to properly link the GORE models with the models of specific MDD approaches.

In general terms, for the application of GORE models to software production processes, the specified models must be manually analyzed to obtain the corresponding software representations. As it is reported in [166], the impossibility of applying requirement models directly into a MDD software production process is due to their nature since these models are centered on problem analysis and not on software representation. Unlike requirement models, the models involved in MDD processes are formulated to provide a precise and complete conceptual representation of the intended software systems in order to achieve automatic software generation by means of model compilations.

Thus, we can conclude that for the appropriate application of GORE modeling to MDD processes, an appropriate input for the model compilation processes must be obtained from the defined requirement models (i.e. to generate an MDD conceptual model from a GORE model).

This chapter presents an approach for automatic linking of GORE modeling and MDD processes by applying the interoperability approach proposed in this thesis. It has been elaborated by taking as reference the experience obtained from the linking of the *i** framework and an the OO-Method MDD, which has been successfully applied to the industrial software development [16]. From this scenario, we show how GORE models can be transformed into the corresponding MDD-oriented models by detailing the following: the customization mechanisms for GORE modeling languages (that are defined to automate the model transformations); the specification of validation mechanisms to assure the

appropriate model transformations; and the generation of required transformation rules.

The rest of this chapter is organized as follows: Section 9.2 presents a background related to benefits of using GORE modeling in MDD processes. This section also shows a general vision of how GORE modeling and MDD processes can interoperate by using the proposed interoperability approach. Section 9.3 details the application of the interoperability approach to link *i** modeling and the OO-Method MDD approach. Section 9.4 presents an analysis of the developed interoperability scenario. Finally, Section 9.5 presents our conclusions.

9.2. Background

Appropriate requirement capturing and elicitation is one of the most important activities in software development, thus the relevance of requirements engineering (RE) to obtain a sound software engineering process. RE clarifies what users want, how they are going to interact with the system, and how the system impacts the business. If these ideas are projected onto the model-driven philosophy, it can be stated that requirement modeling is fundamental in obtaining a sound Model-Driven Engineering (MDE) [13] process for software development. The paper presented in [15] clearly shows the relevance of integrating different modeling approaches to obtain a sound modeling process. This is precisely what we can achieve with the linking of goal-oriented modeling and model-driven development (MDD) processes. .

9.2.1. Improving MDD Processes with *i** Modeling

In the RE domain, the goal-oriented perspective has provided interesting results at both the industrial [164] and research levels [163]. The Goal-Oriented Requirement Engineering is concerned with the use of goals for eliciting, elaborating, structuring, specifying, analyzing, negotiating, documenting, and modifying requirements [165]. In general terms, it focuses on obtaining the 'whys' of the intended systems through the analysis of organizational scenarios. The work presented in [167] shows the relevance of using scenarios for goal modeling, what provides the background for the RE modeling approach considered in this chapter.

Among existing GORE approaches, the *i** framework [68] is currently one of the most widespread modeling and reasoning frameworks [162-164] and it is also well documented [168]. It emphasizes the analysis of strategic relationships among organizational actors capturing the intentional requirements. The term actor is used to generically refer to any unit for which intentional dependencies can be ascribed. Actors are intentional in the sense that they do not simply carry out activities and produce entities, but they also have desires and needs.

The *i** framework [18] captures the intentions behind software requirements using strategic relationships among actors. The *i** framework offers two interrelated conceptual models: the *Strategic Dependency* (SD) model and *Strategic Rationale* (SR) model. The SD model is focused on external relationships among actors, which are called dependencies. In the SD model, the internal goals, knowhow, and resources of an actor are not explicitly modeled.

The SR model expands the description of a given actor and all rationales involved on its intentions, providing support for modeling the reasoning of each actor about its intentional relationships. In addition to the dependency relationships that are present in the SD model the SR model provides three new types of relationships. These relationships are the following:

1. *Task-decomposition* links. These links indicate the elements that are necessary to perform a certain task.
2. *Means-end links.* These links indicate when a task is a means to achieve a goal
3. *Contributions links.* These links indicate how a model element can contribute to satisfy a *soft goal.*

9.2.2. General Schema for the Linking Approach

Our proposal for linking GORE modeling and MDD starts from the idea that there are two kinds of models that must be coordinated to represent the modeling needs of different stages of a common development process: GORE models, and MDD models, which represent different aspects of the intended systems at the conceptual level. These models are represented by using modeling languages whose abstract syntax is specified by means of metamodels.

For the coordination of these two models, we assume that it is possible to partially infer an *initial MDD model* from both the information that is represented in a GORE model and from extra information that is added when

necessary. This MDD model generation is possible if constructs of the MDD modeling language can be inferred from constructs of the GORE modeling language. The constructs involved are represented by the metaclasses of the corresponding metamodels.

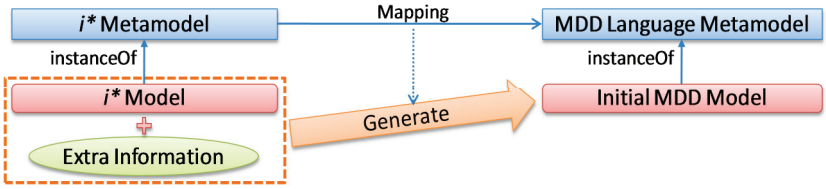

Figure 59. Basic goal-oriented requirements and MDD linking schema

It is important to note that we are referring to an initial MDD model and not a complete MDD model because there are aspects related to specific system functionality that cannot be obtained from requirement models. Therefore, these functional aspects must be specified later, at design time, in the refinement of the initial MDD model that is obtained. Thus, the basic linking schema presented in Figure 59 is the starting point of our proposal. From this initial linking schema, we can state that it is possible to automate the generation of the MDD model by means of well-defined model-to-model transformations, which are based on the metamodels of the modeling languages involved. This automatic generation is possible by using model transformation technologies such as ATL [169] or QVT [59]. However the question of what happens to the required extra information arises. If this extra information is not precisely represented, then the transformation rules cannot be automatically performed. This issue is observed in proposals such as [20] and [170]. In these proposals, guidelines to transform goal-oriented models into software conceptual models are defined, but they must be manually applied because of the lack of a proper mechanism to specify the additional information required.

To solve this problem and to provide a well-defined input for the automatic generation of a MDD model, we use the interoperability approach proposed in this thesis to generate the metamodel extensions that are necessary to represent the extra information that is required. The application of the interoperability approach also generates the necessary interchange information to automatically generate MDD input models from GORE models.

9.3. Applying the Interoperability Approach to *i** and OO-Method

In this section, we explain the different steps of the proposed linking process. Even though this linking process is based on the interoperability approach proposed in this thesis, it requires additional tasks to support the differences in the abstraction level and the application domain of the two involved modeling approaches. In order to facilitate the understanding, we present the process using a brief linking example that is based on the *i** and OO-Method approaches, which correspond to the GORE and MDD counterparts, respectively. Figure 60 shows the *i** diagram related to this example.

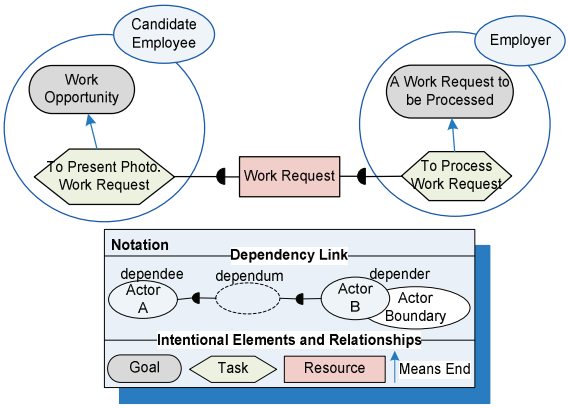

Figure 60. *i** example model

The proposed example represents the reception of work requests (work applications) from potential employees, which is part of a complete case study of a photography agency administration system that was developed in the context of the OO-Method industrial approach (presented in [158]). In order to simplify the example, only a subset of all the *i** and OO-method constructs were used.

9.3.1. Step 1: Definition of the Transformation Guidelines

The first step is to identify those constructs of the GORE modeling approach that are relevant for the generation of constructs of the MDD modeling approach. The identification of the relevant constructs is performed over the metamodels of the modeling languages involved. These metamodels must be EMOF compliant [38] (according to the MDD interoperability process presented in Chapter IV).

Then, the set of transformation guidelines that are needed to obtain the corresponding MDD constructs must be defined from the identified GORE constructs.

For the specification of the involved metamodels, we propose using the Eclipse UML2 tool [134] since it provides automatic generation of EMF metamodels from the defined UML2 metamodels. EMF is the Eclipse Modeling Framework that is based on the EMOF specification. Also, the generated EMF metamodels are tagged with additional information to automatically obtain model editors that have interpreters for the defined OCL rules and that support UML profile extensions.

In the *i** context, there is not a standardized *i** metamodel, and, in general terms, the existing metamodel proposals (such as the one presented in the *i** wiki [171] or in the articles [7, 172]) are not EMOF compliant. However, for the linking example presented here, we can use these proposals as reference for the definition of an appropriate EMOF-based *i** metamodel.

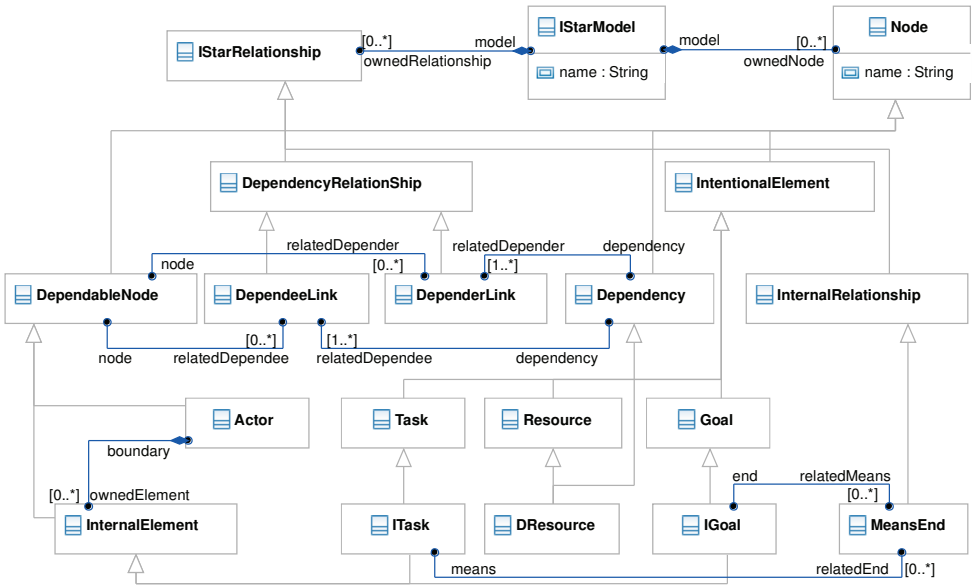

Figure 61. The *i** metamodel for the example model

Figure 61 shows the *i** metamodel defined for the example. In this metamodel, the *i** constructs considered are: actors (class *Actor*); dependency resources (class *DResource*); internal goals and tasks (classes *IGoal* and *ITask*, respectively); and dependency links (class *Dependency*). It is important to note

that this metamodel is only a subset of a complete *i** metamodel. Some of the differences are that *tasks*, *goals*, and *soft goals* can also participate in a dependency link. Therefore, in a complete *i** metamodel, these constructs must be represented as specializations of the class *Dependency* (the same as *DResource*). The *resources*, *goals*, and *soft goals* must also be represented as internal elements (specializations of *Internal Element*) in a complete *i** metamodel. The complete structural representation of the *i** metamodel used as reference for this interoperability scenario is presented in Appendix I.

The OO-Method metamodel used for the proposed example (see Figure 62) is also a subset of the complete OO-Method metamodel.

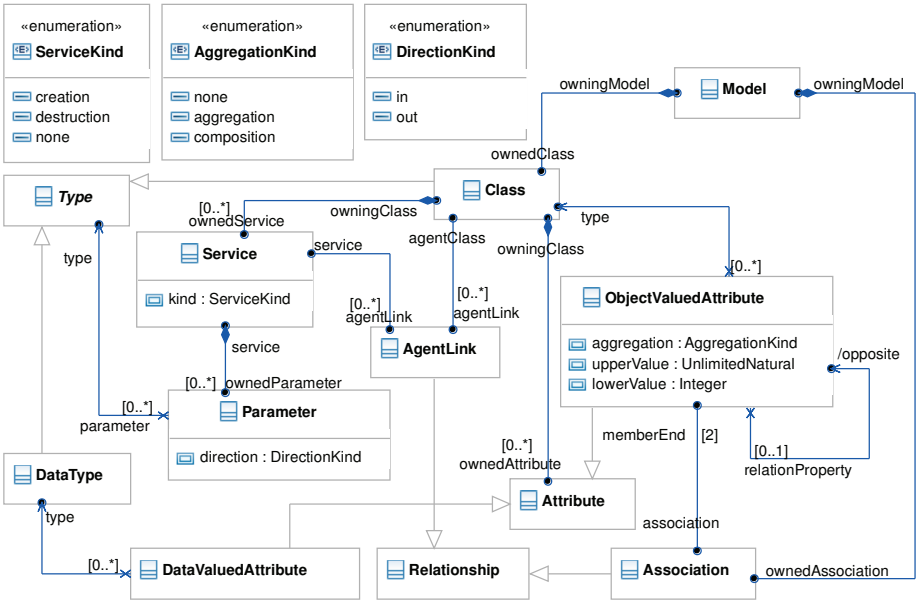

Figure 62. The OO-Method metamodel for the linking example

The presented OO-Method metamodel only includes the essential metaclasses for the definition of classes, attributes, services, associations, and a special relationship that is called *agent link*. This last construct is related to the specification of permissions that a class (of the modeled system) has to execute services of another class. Another particular modeling aspect of the OO-Method class model is the possibility of indicating the services that are capable of *create* or *destroy* instances of the class that owns them. This information is indicated by means of the property *kind*, which is defined in the metaclass *Service*.

Table 9. Guidelines for transformation of *i** models into OO-Method Class Models

*i** Construct	Additional Info	Transformation Guideline
Actor		Class + Agent Link to the Services generated from the actor's internal tasks
Resource	Physical entity	Class
	Informational entity related to a resource or an actor	A Data Valued Attribute that represents information of the Class generated from an actor or a resource
	Informational entity in a resource dependency	A Data Valued Attribute of the Class generated from the dependee actor
Task	Involved in a resource dependency	A Service of the Class generated from the resource
	If it generates a resource	A Creation Service of the Class generated from the resource
Dependency link	Where the dependum resource and the depender and dependee actors are transformed into classes	Associations are automatically defined among the generated Classes

Once the EMOF metamodels are properly specified, the relevant *i** construct must be identified, and the guidelines to transform these constructs into the corresponding OO-Method class model constructs must also be defined. Table 9 shows the transformation guidelines involved in the example. This list is a subset of the transformation guidelines developed for *i** and OO-Method that have been initially presented in [20]. Appendix I shows an improved version of these transformation guidelines and their rationale.

Table 9 also shows the additional information that is required by the transformation guidelines, which may not be present in the *i** metamodel. For instance, an *i** resource is transformed into a class or an attribute depending on whether the resource corresponds to a *physical* or an *informational* entity.

9.3.2. Step 2: Definition of MDD Requirement Metamodel

The second step of the linking process corresponds to properly specifying the modeling information that is required by the transformation guidelines in a format that can be processed by model-to-model transformation technologies [58]. To do this, we define a new EMOF metamodel with the information of the identified *i** elements and the additional information that is required. As a result a specific requirement metamodel for the involved MDD approach is obtained. Figure 63 shows the OO-Method requirement metamodel obtained for the example, which is defined by considering the information presented in Table 9.

In this metamodel, the *i** constructs taken into account are: *actors*, *tasks*, and *resources*.

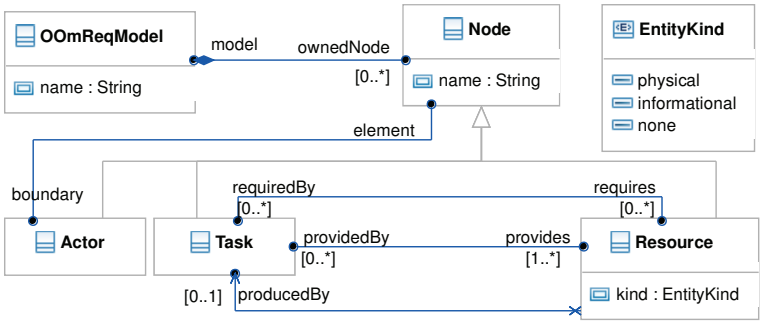

Figure 63. OO-Method requirement metamodel for the *i** linking example

The defined OO-Method requirement metamodel is considerably simpler than the original *i** metamodel, which facilitates the implementation of model-to-model transformations for generating the OO-Method class model. The additional information introduced in this metamodel is the following:

- Specification of resource kind (attribute kind of the metaclass Resource).
- Identification of tasks that generate resources (link *producedBy*).
- Identification of tasks that participate in a resource dependency (links *requiredBy* and *providedBy*).

To name the links involved in a resource dependency, we consider that the task related to the *depender* actor *requires* the resource for its execution, while the task related to the *dependee* actor is responsible for *providing* the resource.

Thus, at the end of the second step, we obtain two metamodels: the original *i** metamodel (the original GORE metamodel) and the OO-Method requirement metamodel for the generation of the OO-Method class model (the MDD requirement metamodel).

9.3.3. Step 3: Definition of Validation Rules

In this step, syntactical validation mechanisms are specified in order to perform a correct generation of the corresponding MDD models. These validation mechanisms must be defined in the MDD requirement metamodel (generated in step 2) since this metamodel has all the information to perform the model transformations. For instance, in the linking example, an *i** resource is

transformed into an attribute or a class, depending on whether the resource is specified as an *informational* entity or a *physical* entity (see Table 9). From this transformation guideline, a possible validation is to assure the appropriate specification of the kind of resource. This validation cannot be specified in the *i** metamodel since the information related to kind of resource is not present.

For the specification of these syntactical validations, we propose the use of OCL rules since OCL is also part of the OMG standards for the specification of metamodels; hence, it is defined to work in conjunction with MOF. In addition, the OCL rules can be automatically processed by tools such as [134]. Thus, for the previous validation example, we can define the following OCL rule in the class *Resource* of the OO-Method requirement metamodel:

Context: Resource::ValEntityKind()

Body: self.kind = Physical or self.kind = Informational

It is important to note that the modeling information that is not present in the original GORE metamodel is the critical point to be validated for the correct generation of the MDD model for the following two reasons:

1. The modeling information that exists in the GORE metamodel has probably already been validated
2. The new modeling information is essential for performing the model transformations, and hence, an incorrect specification of this information will produce an incorrect generation of the MDD model.

9.3.4. Step 4: Application of the Interoperability Approach

The fourth step of the linking approach is to go from the models that are based on the original GORE metamodel to the specific requirement models for the MDD approach that are based on the MDD requirement metamodel. This is because the intention of the linking proposal is to use the original GORE modeling approach for requirement modeling. In the example, this corresponds to going from *i** models that are based on the original *i** metamodel (Figure 61) to requirement models that are based on the generated OO-method requirement metamodel (Figure 63).

However, this step is not trivial since the additional modeling information and validation rules that are present in the defined MDD requirement metamodel are not present in the original GORE metamodel. Thus, in this step, the interoperability approach (see Chapter IV) is put into practice to obtain the

required metamodel extensions for the GORE metamodel and the needed model interchange information. Figure 64 shows the resultant linking schema with the different input and output elements that are related to each step (numbered from E1 to E9).

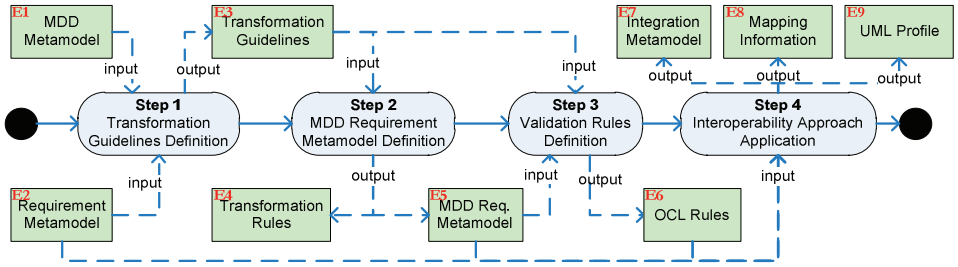

Figure 64. General Schema of the proposed linking process

Figure 64 shows that Step 4 of the process generates the corresponding Integration Metamodel and UML Profile, as well as, mapping information (among the metamodels involved) for the automatic interchange of models.

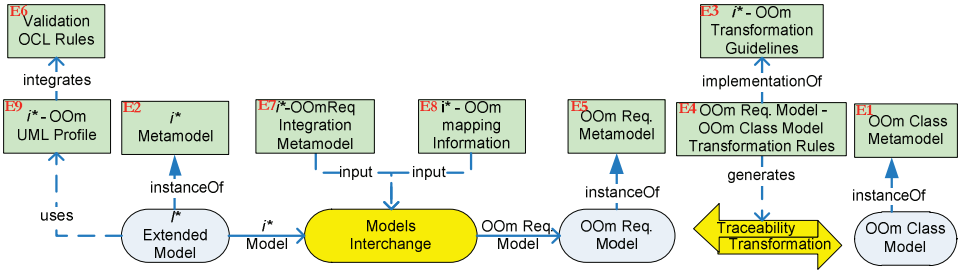

Figure 65. Interoperability proposal applied to *i** and OO-Method

Figure 65 shows how each one of the input and output elements considered in the interoperability process are used to link the *i** framework and the OO-Method MDD approach. This figure also shows the generation of traceability information [173, 174], which is necessary to maintain the relationships between the software specification (described in the MDD model) and the requirement specification (described in the MDD requirement model). The generation of this traceability information must be implemented together with the transformation rules for the MDD requirement model.

Figure 66 shows the Integration Metamodel obtained for the example. This metamodel is generated from the OO-Method requirement metamodel by

applying the systematic approach presented in Chapter V. This systematic approach is based on taking the OO-Method requirement metamodel (the source metamodel) and performing a set of redefinitions over it to align this source metamodel to the structure of the *i** metamodel (target metamodel). This redefinition allows the automatic identification of the extensions that are required to introduce the modeling needs of the source metamodel into the target metamodel, that is, to extend the *i** framework to represent the information of the OO-Method requirement model.

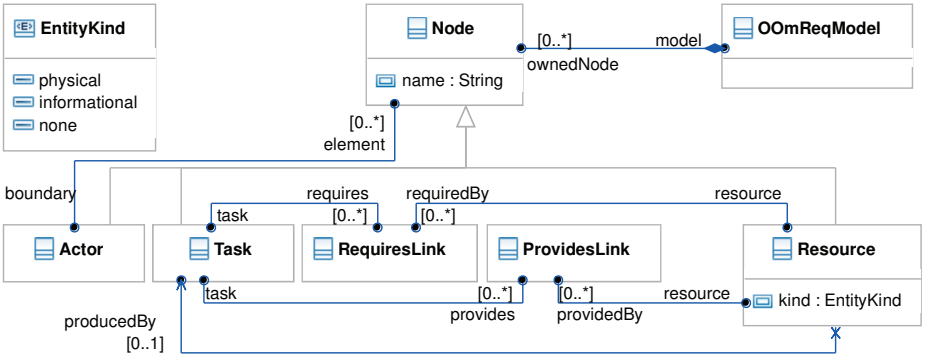

Figure 66. Integration Metamodel for the integration example

The resultant Integration Metamodel shows the classes *AffectsLink* and *RequiresLink*, which are not present in the OO-Method requirement metamodel. These classes are defined to perform the correct mapping from the associations *task.requires* and *task.provides* (which are derived from dependency links) to the *i** constructs *DependeeLink* and *DependerLink*. This is done since the mapping can only be performed among elements of the same kind (classes with classes, associations with association, and so on) [127].

There are four conditions that an Integration Metamodel must hold for the automatic generation of the metamodel extensions. These are the following:

- All the classes from the Integration Metamodel are mapped to the target GORE metamodel. This assures that the constructs from the MDD requirement metamodel can be represented from the constructs of the GORE metamodel. Table 10 shows the mapping obtained for the linking example.
- The mapping is defined between elements of the same type (classes with classes, attributes with attributes, and so on).

- An element from the Integration Metamodel is only mapped to one element of the GORE Metamodel.
- If the properties (attributes and associations) of a class A from the Integration Metamodel are mapped to properties of a class B of the GORE metamodel, then the class A is mapped to the class B or a specialization of it.

Table 10. Integration Metamodel and the *i** metamodel mapping

I.M. Element	*i** Element	I.M. Element	*i** Element
Node	Node	Resource	DResource
.model	.model	.kind	(No equivalence)
.name	.name	.providedBy	.relatedDependee (inherited from Dependency)
.boundary	.boundary	.requiredBy	.relatedDepender (inherited from Dependency)
OOmReqModel	IStarModel	.producedBy	(No equivalence)
.name	.name	Task	ITask
.ownedNode	.ownedNode	.provides	.relatedDependee (inherited from DependableNode)
Actor	Actor	.requires	.relatedDepender (inherited from DependableNode)
.element	.ownedElement	RequiresLink	DependerLink
ProvidesLink	DependeeLink	.task	.node
.task	.node	.resource	.dependency
.resource	.dependency	EntityKind	(No equivalence)

By applying the automatic UML profile generation to the Integration Metamodel defined (see Chapter VI), the corresponding UML profile that implements the required *i** extensions is obtained (see Figure 67). In the generated UML profile, the properties that have no equivalence in the target *i** metamodel are defined as new properties (tagged values) in the stereotypes that extend the metaclasses.

In the Integration Metamodel definition and the UML profile generation, specific mappings among the participant metamodels are generated, which are used to perform the automatic transformation of GORE models into MDD requirements models. These mappings are the following:

1. The mapping between the Integration Metamodel and the extended GORE metamodel. In the example, this corresponds to an extended version of the mapping presented in Table 10, where the elements of the Integration Metamodel that have no equivalence in the *i** metamodel are mapped to the corresponding UML profile elements.
2. The mapping between the MDD requirement metamodel and the Integration Metamodel. Table 11 shows the mapping obtained for the linking example.

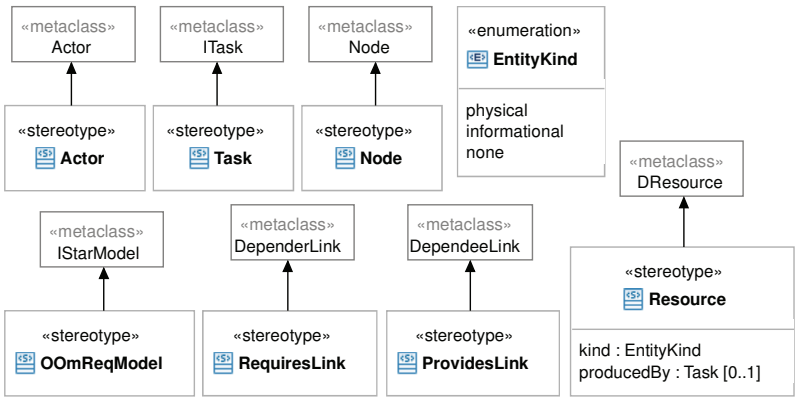

Figure 67. UML Profile generated from the Integration Metamodel of the example

Table 11. OO-Method requirements metamodel and Integration Metamodel mappings

OO-Method Req. Element	I.M. Element	OO-Method Req. Element	I.M. Element
Node	Node	Resource	Resource
.model	.model	.kind	.kind
.name	.name	.providedBy	.providedBy.task
.boundary	.boundary	.requiredBy	.requiredBy.task
OOmReqModel	OOmReqModel	.producedBy	.producedBy
.name	.name	Task	Task
.ownedNode	.ownedNode	.provides	.provides.resource
Actor	Actor	.requires	.requires.resource
.element	.element		

Finally, the OO-Method class model presented in Figure 68 is obtained from the example *i** model that is extended with the generated UML profile. In the

extended *i** model (see Figure 68), we considered the resource *Work Request* as a *physical* entity produced by the task *To Present Work Request*.

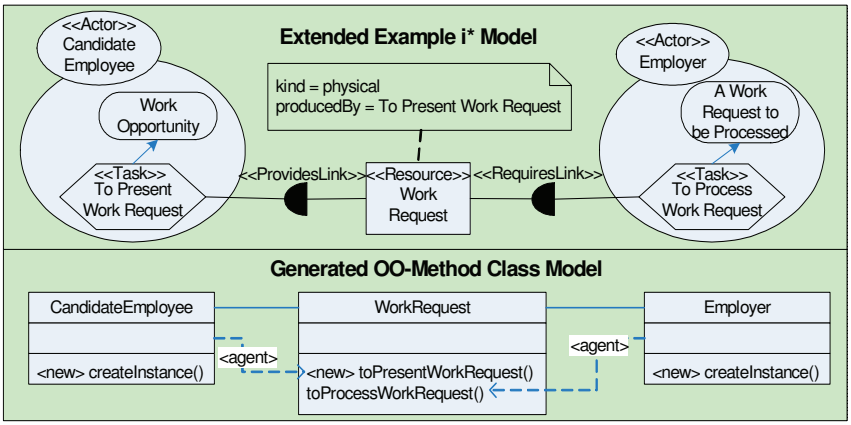

Figure 68. Extended example *i** model and the OO-Method class model generated

The generation of the OO-Method class model is performed by means of model-to-model transformation rules that are defined according to the interchange proposal presented in Chapter VII, which is driven by the metamodel mappings presented in Table 10 and Table 11, and from the transformation guidelines presented in Table 9.

Figure 68 shows that the *i** actors are transformed into classes. The same occurs for the resource *Work Request* since it is a physical entity. The agent relationships are also represented to indicate the permissions that the classes *CandidateEmployee* and *Employer* (generated from the corresponding *i** actors) have over the services of the class *WorkRequest,* which were generated from the defined *i** tasks. The task *to Present Work Request* is transformed into a creation service of the class *WorkRequest* since this service *generates* this resource. The creation services are identified by the tag <*new*> (inferred from the property *kind* of the metaclass *Service* of the OO-Method metamodel). In addition, during the generation of the class model, a creation service is automatically generated for the classes *CandidateEmployee* and *Employer* since, in OO-Method, all classes must have at least one creation service.

Figure 68 also shows that the generated class model has no attribute definition or arguments for the services since this modeling information cannot be derived from the example *i** model. The same happens with the functional specification of the generated services. Therefore, this information must be

specified at the design stage in order to generate a complete class model from the initial class model generated. Thus, from the complete model, the final executable application can be automatically obtained through the *OO-Method model compiler* [12].

Figure 69 shows a graphical example of how the transformation of the *i** model is performed. This example shows the transformation of the resource *Work Request* and the task *To Present Work Request* to the corresponding constructs of the OO-Method class model. It is important to note that this transformation is automatically performed by means of the transformation rules; hence, the generation of the intermediate models is transparent. These intermediate models are the instances of the Integration Metamodel and the MDD requirement metamodel.

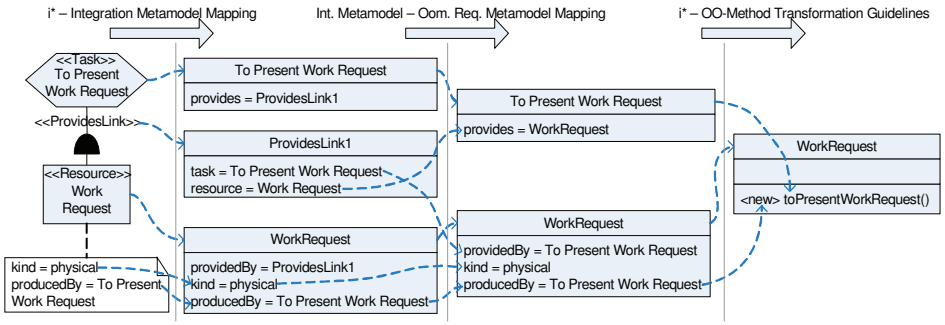

Figure 69. Transformations to obtain an OO-Method class model from an *i** model

9.4. Analysis of the Proposal and Discussion

In the literature, there are papers that are oriented to generating conceptual models from GORE models. However, most of these papers are based on standard UML models (such as [161]), and, in general terms, UML does not offer all the modeling information necessary to participate in an effective MDD process [31]. Furthermore, most of the works that are oriented to go from GORE models to more specific design models, such as [170, 175-177], are not based on standards or well-defined processes, nor do they introduce automation possibilities. Therefore, the application of these proposals must be manually performed [67]. This is not a suitable option because the manual translation of models is a time consuming and error prone task [168]. Hence, automatic linking

of GORE models and MDD approaches takes on special relevance for the adoption of new development paradigms and the improvement of development processes.

One important aspect that must be discussed about our proposal is how to identify the subset of GORE modeling constructs that must be considered for the generation of MDD models since it is very probable that not all the elements of the defined *i** model have to be considered for the development of a software product. In the proposal, even though the constructs that participate in the MDD model generation are identified, this is not enough to assure that only the elements that are related to the software specification participate in the transformation. For instance, in the example, the *i* Actor* is considered in the class model generation, but in a real *i** model some actors may not be relevant for the intended system, and, therefore, they must not be transformed into classes of the class model. UML profiles provide a suitable solution for this issue since it is possible to indicate that only those stereotyped (extended) elements must be considered in the transformation process. This is an important reason for using UML profiles instead of other metamodel extension mechanisms [33]. Other reasons are that the UML profile has a standard specification [131] and a standardized interchange format (XMI [40]).

Another interesting discussion point of this proposal is the need for defining an Integration Metamodel instead of a direct mapping between the original GORE metamodel and the MDD requirement metamodel. The definition of an Integration Metamodel is performed because a direct mapping does not always provide enough information to automatically identify the required metamodel extensions [18, 135]. Also, a direct transformation is dependant on the extension mechanism selected. In contrast, the Integration Metamodel allows the required extensions to be automatically identified independently of their final implementation.

Some additional benefits of the Integration Metamodel are the following: it automates the generation of the required transformation rules; the required extensions can be validated before its implementation; it allows the automatic generation of the mapping for the interchange of models; and it provides a common interface between the GORE metamodel and the MDD requirement metamodel. This last benefit prevents a change in the original GORE metamodel from affecting the transformation rules that are defined in the MDD requirement metamodel. These benefits are better perceived in real GORE models that are more complex than the example presented.

The Integration Metamodel is also useful for MDD approaches that already have a requirement modeling approach. In this case, the MDD requirement metamodel is the metamodel of the existing requirement approach. The next steps of the process are normally applied over this metamodel, and the differences that may exist with the target GORE metamodel (for instance, the *i** metamodel) are managed by the Integration Metamodel and the metamodel extensions.

9.5. Conclusion

In this chapter, an interoperability scenario related to link GORE models and MDD approaches has been developed. This linking is performed by means of the proposed MDD interoperability process, which is oriented to obtaining the mechanisms for automatic generation of MDD-oriented conceptual models from GORE models. The interoperability approach presented in this thesis provides a suitable solution to take advantage of existing standards and technologies, which facilitates the application of the presented proposal to different MDD approaches. In addition, existing open-source tools, such as [58, 126, 134], can be used to implement the required metamodels and model transformations.

Nevertheless, it is very difficult to find requirement editors that support the standards that are considered in this proposal. For instance, we have not found an *i** editor that is compatible with the MOF specification or that supports modeling extensions, in spite of this chapter shows the relevance of requirement technologies that provide extension facilities to obtain an appropriate linking with MDD approaches. Hence, we believe that appropriate requirement modeling tools that are aligned with the capabilities provided by the current standards and technologies for the specification of modeling languages should be implemented.

In the next chapter a very relevant aspect of the interoperability in MDD processes is analyzed. This is the need of assure the models defined with the customized modeling languages are correctly defined for their transformation into the corresponding MDD-oriented models. Therefore, a proposal for automatic verification of the models involved in the MDD interoperability has been defined. This proposal can be considered as an alternative to perform the third step of the linking process defined in this chapter, which corresponds to the definition of validation rules for the transformation of the *i** models.

Chapter X.
Automatic Verification of Models for MDD Interoperability

From the application of the interoperability scenarios developed in this thesis, we have detected that the involved models can present issues that prevent an appropriate transformation, and, hence, an incomplete interchange of the modeling information is performed. Furthermore, in the context of the i and OO-Method interoperability, we have detected that the input requirement models can be improved by fixing these interoperability issues. Thus, a more complete requirement specification is obtained, which also produce a more complete generation of the corresponding MDD models.*

Thus in this chapter we present a proposal to guarantee the correct MDD interoperability by means of the integration of verification mechanisms into the involved modeling language. In particular, we present an approach based on the i framework and OO-method interoperability scenario. This verification proposal is based on a specific process for the definition and implementation of i* verification measures.*

The results obtained from the application of the verification approach are empirically validated to determine if the verification measures assure the completeness of the i model transformations in the MDD interoperability context.*

10.1. Introduction

The present software development context is rapidly moving to the Model-Driven Development (MDD) paradigm [1], which has motivated the emergence of multiple MDD approaches oriented to automating the final software product generation by means of model compilation processes. Just as any software development process does, MDD processes also require an appropriate requirement elicitation activity to obtain software products that fit the customers' needs. Integrating the requirements elicitation activity into the MDD processes, it should be possible to obtain software products properly aligned with the stakeholders' needs [178]. In addition, it should be possible to estimate the impact that the generated software systems have in the organizational objectives, and to identify different alternatives for the configuration of the intended software systems.

Among modeling approaches to requirement elicitation, the *i** framework [6] provides a suitable alternative for the analysis of complex scenarios. The *i** framework is a goal-oriented [179] approach used in several activities and contexts of software engineering, and in particular, in the early phases of requirements engineering [69]. The versatility and expressive power of *i** is extensively documented [180]. Thus, we consider that it is a good choice for the requirement elicitation in MDD processes.

To achieve this goal, we have applied the MDD interoperability approach presented in this thesis, which corresponds to the interoperability scenario presented in the previous chapter (Chapter IX). However, we have detected that input *i** models may have modeling issues that prevent an appropriate model transformation, thus producing an incomplete interchange of the involved modeling information. Thus, to apply our interoperability approach to real development contexts, additional verification mechanisms are required to assure the correct specification of the involved models. This idea is applicable not only to the *i** and OO-method interoperability scenario, but also, to any interoperability scenario where automatic transformation of the involved models is necessary. Thus the goal of this chapter is present a verification approach to assure the correct MDD interoperability execution. This verification approach is presented and explained by using the *i** and OO-Method interoperability as reference. In particular, we present our proposal by considering the following three element:

1. **Definition:** Presents a specific proposal for the systematic definition of *i** verification measures, which assure the completeness of the MDD models that are generated from *i** models. Thus, the verification measures act as indicators of modeling issues, which identify the *i** elements that need to be fixed to assure the automatic generation of input models for MDD processes.
2. **Integration:** The three first steps of the proposed MDD interoperability process are applied to integrate into the *i** framework the defined verification measures and the modeling information that is necessary to automatically transform *i** models into MDD models.
3. **Evaluation**: The verification proposal is empirical validated through a laboratory experiment, which demonstrates that the measures obtained provide support to achieve the completeness of the generated MDD model. The execution of this experiment is also used to show how the verification measures are applied to improve *i** models in the OO-Method MDD context.

The rest of the chapter is organized as follows. Section 10.2 shows the main concepts and elements that are involved in the definition of the verification proposal. Section 10.3 details the proposed verification process. Section 10.4 explains how the verification measures can be used to fix and improve *i** models. Section 10.5 presents an empirical study performed to evaluate the efficacy of the verification proposal. Section 10.6 presents an overall analysis of the proposal. Finally, Section 10.7 presents our conclusions.

10.2. Background

In this section, the main metrology concepts used in this chapter are clarified. Afterwards, the relevance of applying verification mechanisms for the application of requirement models in MDD processes is briefly discussed.

10.2.1. Clarifying Verification and Measure Concepts

In the literature, there is no consensus for the concepts used in the software measurement field. This has provoked that different concepts are used to refer to the same things, or even the same concept is used to refer to different things. Thus, we have carefully analyzed the measurement standards to properly use the

terms involved in this chapter, which are related to the concepts of verification and measure.

Verification is defined in the International Vocabulary of Basic and General Terms in Metrology [181] as "*confirmation through examination of a given item and provision of objective evidence that it fulfills specified requirements*". In contrast, validation is defined in the same standard vocabulary as "*confirmation through examination of a given item and provision of objective evidence that it fulfills the requirements for a stated intended use*". These definitions are also agreed with the widely used Barry Boehm's definitions to verification (doing the system right) and validation (doing the right system) [182]. Thus, we use the term verification instead of validation since we focus in the correct application of the transformation guidelines defined to go from *i** models to MDD-oriented models.

Even though the most of referenced works related to software measurement use the term metric instead of measure, we use the term measure because it is more appropriate to the objectives of our approach (we measure *i** elements) and it prevents ambiguous interpretations. This term distinction is clearly presented in the paper [183]. Extending the concept of measure we have introduce in this paper the term *verification measure*.

It is important to clarify that a *verification measure* is not referring to a verification mechanism by itself; it is a special measure that supports the verification and improvement of an *i** model by means of a proper analysis of the information reported.

10.2.2. Requirement Models and MDD Processes

As we can observe in the systematic review about requirement engineering and MDD presented in [184], several approaches (such as [84, 161, 178, 185]) have encouraged the use of high-level analysis models (i.e., requirement models) as part of a sound MDD process. A representative example is the MDA approach [10], which proposes the definition of a *Computation-Independent Model* as starting point of the development process [186]. However, most of the current requirement approaches are not automatically applied, or are not based on modeling standards [187]. Thus, an effective solution that includes requirement models as part of a complete, standardized, and automatic MDD process [188] is still an unsolved challenge [184].

Probably, one of the main issues to achieve this requirement modeling and MDD linkage is the proper definition of the requirement models for the automatic generation of domain-specific models [3] related to MDD processes. Most of the proposals oriented to translate requirement models into MDD models (such as [189] and [190]) are considering the input requirement models to be properly defined to perform the translation. We know this idealist scenario is not applicable in practice, and verification mechanisms are necessary to assure the generation of the corresponding MDD models.

To assure the automatic requirement transformation, certain proposals suggest the manual translation of the defined requirement documents to a specific computable format [185, 191]. These approaches restrict the flexibility of the original specification, which, together with the manual translation of the requirements, may cause loss of information.

Other approaches suggest to add quantitative information to existing requirement modeling approaches [192-194], which allows the automatic measure and analysis of the defined models without restricting their original specification. However, there is a lack of measures to support the verification of requirement models for generation of domain-specific models [3] related to MDD processes. Hence, to fill this gap, we have considered the approaches related to object-oriented models verification [195, 196], and definition of measures to verify the correct compilation of domain-specific models [197].

10.2.3. The Photography Agency *i** Model

For the presentation of the proposed verification approach, we have used a detailer version of the model presented in the *i** and OO-Method interoperability scenario (see Chapter IX), which is related to the management of work requests in a Photography Agency (see Figure 70). This *i** model is defined from the OO-Method study presented in [158]. In order to simplify the *i** model representation, the soft goals are omitted in the example since this *i** construct does not participate in the generation of the target MDD models. Even though a similar situation occurs for *i** goals, this constructs are represented to be consistent with the *i** framework notation (see Figure 6 1.1 from the background chapter). The organizational description related to the example *i** model definition is presented below:

The photography agency is dedicated to the management of photo reports and their distribution to publishing houses. This agency operates with freelance

photographers, who must present a request to the production department of the photography agency. This request contains: the photographer's personal information, a description about the equipment owned, and a brief curriculum vitae. An accepted photographer is classified by the production department in one of three possible levels for which minimum photography equipment is required. The possible levels are defined by the commercial department, who establishes the price that will be paid to the photographer and the price that will be charged to the publishing house for each photo.

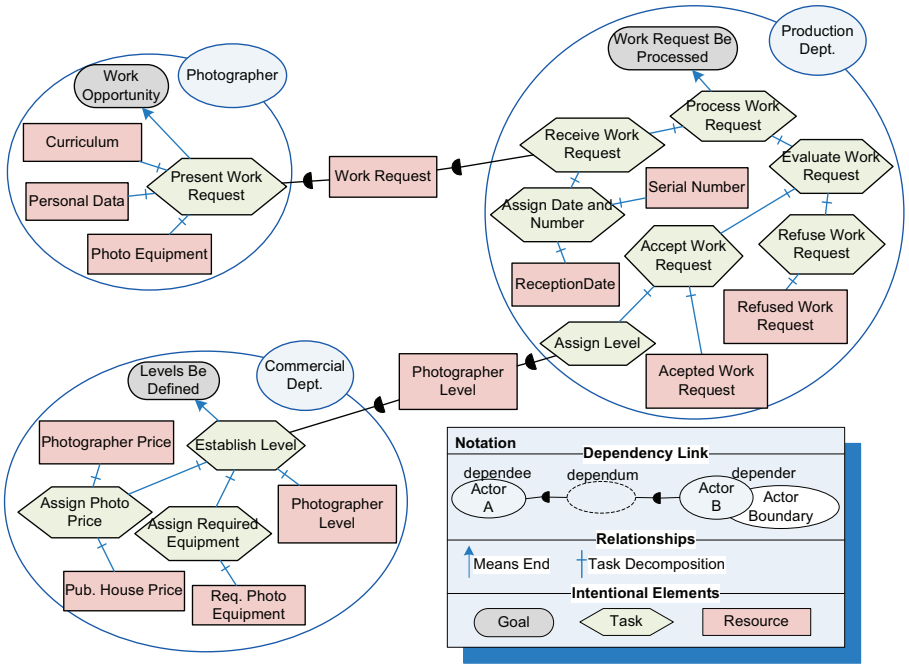

Figure 70. Example *i** SR Model.

In general terms, the presented *i** model shows how the *production department* depends on the reception of *work requests* (i.e. job applications) that are produced by *photographers* that want a *work opportunity*. The *work requests* are comprised by the photographer's *personal data*. The *production department* is the responsible for *refusing* or *accepting* the *received work requests* by indicating the final *work request status*. For the accepted requests a *photographer level* is assigned according to the information provided by the *Commercial Department*.

10.2.4. Transformation of *i** Models

According to the *i** and OO-Method interoperability scenario, a set of transformation guidelines must be defined to generate appropriate MDD models from the defined *i** models. Table 12 summarizes a representative subset of transformation guidelines (adapted from [20]) for *i** and OO-Method, which have been selected due to their applicability to other MDD approaches based on class model specification. These guidelines are used to exemplify the proposed verification approach throughout this chapter. The rationale of these guidelines is presented in Appendix I. Table 12 shows the *i** constructs that are involved in the transformation, the additional information that is required to perform the transformation and the target constructs of the class model.

Table 12. Guidelines for the transformation of *i** models into OO-Method class models

*i** Construct	Additional Information	Class Model Construct
Actor		Class
Resource	Physical entity	Class
	Informational resource related to a physical resource or an actor	An attribute of the class generated from the actor or physical resource
	Informational resource inside of an actor boundary	An *agent relationship* between the class generated from the actor and the attribute generated from the resource
Task	If generates an entity (physical resource or actor)	An instance creation service of the class generated from the corresponding entity
	If affects the state of a resource	A service of the class generated from the resource or from the owner physical resource.
	If does not affect resources or generate entities	A service of the actor that contains the task
	If is decomposed in resources	Associations are automatically defined among the class that contain the corresponding service and the classes generated from the decomposed resources
	Inside of an actor boundary	An *agent relationship* between the class generated from the owner actor and the task
Resource Dependency Link		Associations are automatically defined among the class generated from the *dependum* resource and the classes that own the services generated from the involved tasks
Is-a Link		A generalization relationship is generated between the classes generated from the involved actors

The guidelines presented in Table 12 can be combined, for example, a physical resource that is a *dependum* in a dependency link generates a class and associations among this class and the classes that own the services generated from the tasks involved.

For the transformation guidelines related to tasks and dependency links, when the resource involved corresponds to an informational resource, the rule is applied to the physical resource related to the informational resource. For instance, a task that affects the state of an informational resource is transformed into a service of the class generated from the physical resource that owns the attribute generated from the informational resource.

In the transformation guidelines presented in Table 12 it is possible to observe a specific OO-Method construct: the *Agent Relationship*. This construct corresponds to a binary relationship that indicates the visibility and execution permissions that a class of the model has over other classes or over itself (recursive agent relationship). The classes that have agent relationship to other classes are named *Agents* of the modeled systems. This construct is relevant for the specification of interaction models, such as the OO-Method presentation model (see Section 1.1 from the background chapter). Even though the agent construct is specific for OO-Method, its semantics can be generalized to other MDD approaches that define system users and interaction aspects at conceptual level.

Thus, for the automatic application of the transformation guidelines it is important to determine if the defined *i** models provide a proper specification, and, hence, the verification of the *i** models becomes necessary.

10.3. Integration of Verification Measures into the *i** Framework

This section explain the process for the definition and integration of verification measures into the *i** framework. For the elaboration of this process, we have considered existing standards and modeling technologies to facilitate its application for different MDD approaches. The technologies and standards involved are: approaches for the specification of measures [183, 198, 199], the last version of the *i** framework [171], approaches for the definition of *i** measures [200, 201], OMG Standards for metamodeling [38] and model

extensions definition [51], and Eclipse Model Development Tools [134]. The steps of the process are described below (see Figure 71).

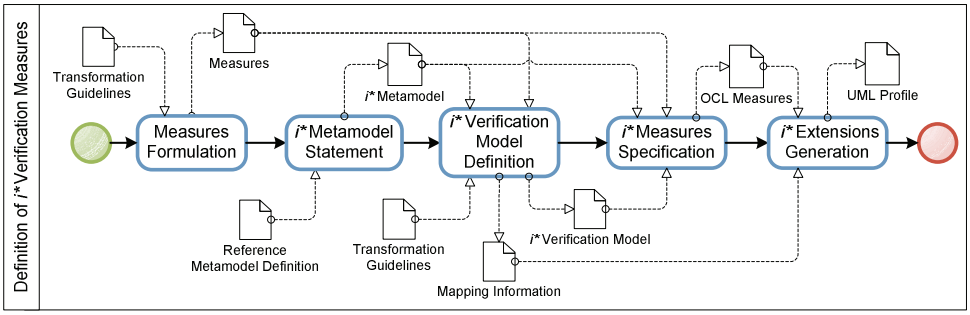

Figure 71. Process for definition of *i** verification measures.

10.3.1. Step 1: Measures Formulation.

The first step of the process considers the appropriate formulation of the *i** verification measures. This means identifying the *i** constructs that participate in the MDD model generation, and, from these, identifying the aspects that must be verified for a correct *i** model transformation.

To perform the identification of the involved *i** elements, it is necessary to know the transformation guidelines (or rules) related to the target MDD model generation. In particular, we focus on the additional information that must be specified by the analyst to perform the corresponding transformation, which is the critical point that must be verified to assure that the transformation can be performed correctly and automatically.

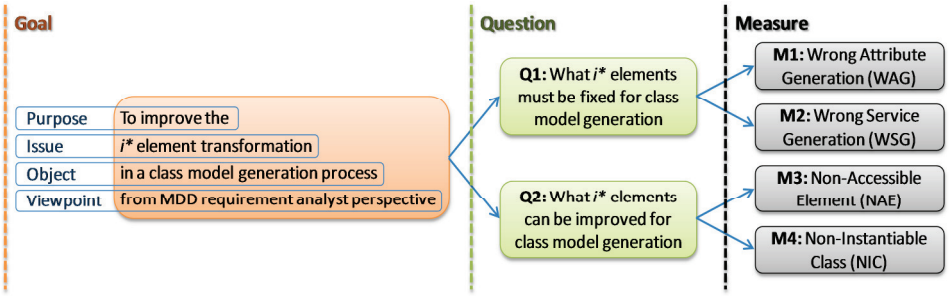

Figure 72. Application of the GQM approach.

The measures formulation is performed by applying the Goal-Question-Metric (GQM) approach [198]. Figure 72 shows an excerpt of the application of the GQM approach to the transformation guidelines presented in Table 12. For the formulation of the questions related to the GQM approach, we suggest to consider two verification levels. These levels are related to the following elements:

1. The *i** elements that must be necessarily fixed because they cannot be transformed or produce a wrong class model generation (i.e., Q1 in Figure 72).
2. The *i** elements that can be correctly transformed, but they still can be improved to obtain a more complete class model generation (i.e., Q2 in Figure 72).

It is important to consider that the verification measures formulated in this step are specific for the transformation guidelines presented in Table 12. Thus, other MDD approaches with different transformation guidelines will require different (or additional) verification measures. However, since we have intended to select a representative set of transformation guidelines that can be generalized for MDD-oriented class model generation, the resultant measures can provide relevant verification information to other object-oriented MDD approaches.

The measures that are related to answer each of the presented GQM questions are specified by considering the framework presented in [183]. This framework specifies that the empirical world, the numerical world, the measurement method, and the measurement procedure must be defined in the design of measures. In the definition of the empirical world, the entity and the attributes to be measured must be identified. An *entity* corresponds to an input artifact used to perform a measurement, and the concepts related to that artifact correspond to measurable *attributes*. In the definition of the numerical world, the measurement *scale* must be defined. In the specification of the measurement method, the *measurement principle* must be identified. Finally, in the specification of the *measurement procedure*, details of the application of the measurement method must be defined. This measure specification framework is applied to the four measures previously formulated.

M1. Wrong Attribute Generation (WAG)

Rationale. The informational resources are involved in the generation of class attributes (see Table 12). Therefore, for the correct generation of informational resources, they must be related to a system entity (actor or a physical resource),

which is transformed into a class in the class model. Otherwise, it is impossible to transform these resources into attributes because the class that contains them cannot be identified. Table 13 details the characteristics related to this measure according to the considered definition framework.

Table 13. Characteristics of measure Wrong Attribute Generation (WAG)

Characteristic	Definition
Measurement Entity	*i** model
Measurement Scale	Ratio scale
Attribute to be measured	Informational resources not related to a physical resource nor to an actor
Measurement principle	An informational resource that is not related to a physical resource nor to an actor is directly proportional to a wrong attribute generation in the MDD model
Measurement procedure	The attributes to be measured must be counted to obtain the number of informational resources that cannot be transformed into attributes

Following, the formula to obtain the measure M1 – WAG is presented:

$$WAG_M = \sum_{\substack{r \in resources(M) \wedge \\ kind(r) = Informational}} conv(\neg relatedToActor(r) \wedge \neg relatedToPhysResource(r)) \quad Conv(x) \begin{cases} 1, \text{if } x = true \\ 0, \text{if } x = false \end{cases}$$

M2. Wrong Service Generation (WSG)

Table 14. Characteristics of measure Wrong Service Generation (WSG)

Characteristic	Definition
Measurement Entity	*i** model
Measurement Scale	Ratio scale
Attribute to be measured	Tasks that not generate entities nor affect resources and the related actor is not marked for the generation of the intended system
Measurement principle	A task that not generates entities nor affects resources and it is related to an actor not marked for the generation of the intended system is directly proportional to wrong service generation in the generated MDD model
Measurement procedure	the attributes to be measured must be counted to obtain the number of wrong services specified in the generated MDD model

Rationale. According to the transformation guidelines, the tasks that do not generate entities (physical resources or actors) or that do not affect resources are

transformed into services of the class generated from the owner actor (according to the corresponding actor boundary). Therefore, if the corresponding actor is not marked for the generation of the intended system, the involved task cannot be transformed since it is not possible to generate a service in the class model without a class that contains it. See WSG characteristics in Table 14.

Following, the formula to obtain the measure M2 – WSG is presented:

$$WSG_M = \sum_{t \in tasks(M)} conv(\neg generatesResource(t) \wedge \neg affectsResource(t) \wedge \neg hasSystemActor(t))$$

M3. Non-Accessible Element (NAE).

Rationale. According to the presented transformation guidelines (see Table 12), agent relationships are defined between the classes generated from actors and the elements generated from services or informational resources contained in the corresponding actor boundaries. However, if the involved actors are not selected for the MDD model generation, the actor is not transformed in a class, and, hence, the involved agent relationships are not defined. This produces that the transformed tasks or informational resources (that are inside of the actor boundary) cannot be executed or visualized in the final application.

Table 15. Characteristics of measure Non-Accessible Element (NAE)

Characteristic	Definition
Measurement Entity	*i** model
Measurement Scale	Ratio scale
Attribute to be measured	Internal tasks or resources related to the system that are defined in the boundary of an actor that is not related to the system
Measurement principle	An internal task or resource related to the system that is defined in the boundary of an actor that is not related to the system is directly proportional to the a non-accessible element in the generated MDD model
Measurement procedure	the attributes to be measured must be counted to obtain the number of non-accessible elements in the generated MDD model

However, it is not mandatory to define an actor as part of the intended system. For instance, the analyst could consider that the involved actor must not be maintained in the final system. In this case, a new agent (special user) must be defined at design time during the refinement of the generated class model to execute and visualize the generated elements, such as an administrator user. See

NAE characteristics in Table 15. Following, the formula to obtain the measure M3 – NAE is presented:

$$NAE_M = \sum_{t \in tasks(M)} conv(\neg hasSystemActor(t)) + \sum_{\substack{r \in resources(M) \wedge \\ kind(r)=Informational}} conv(\neg hasSystemActor(r))$$

M4. Non-Instantiable Class (NIC)

Rationale. The system entities (physical resources or actors) without a production task related are transformed into classes without an instance-creation service (see Table 12). The service that produces new instances of a class takes special relevance since without this service, the class is not properly defined (all the defined classes must be capable of generating their instances). However, the definition of production tasks for entities (actors or physical resources) is not mandatory since this issue does not prevent the appropriate transformation of the *i** elements. Thus, specific instance-creation services can be defined at design time for the classes generated without this kind of services. See NIC characteristics in Table 16.

Table 16. Characteristics of measure Non-Instantiable Class (NIC)

Characteristic	Definition
Measurement Entity	*i** model
Measurement Scale	Ratio scale
Attribute to be measured	Actors and physical resources without a task related to their production
Measurement principle	An actor or a physical resource without a related production task is directly proportional to a non-instantiable class in the generated MDD model.
Measurement procedure	The attributes to be measured must be counted to obtain the number of non-instantiable classes

Following, the formula to obtain the measure M4 – NIC is presented:

$$NIC_M = \sum_{\substack{r \in resouces(M) \wedge \\ kind(r)=Physical}} conv(\neg hasProductionTask(r)) + \sum_{a \in actors(M)} conv(\neg hasProductionTask(a))$$

10.3.2. Step 2: *i** Metamodel Statement

The second step corresponds to stating the target *i** metamodel, which must be defined according to the EMOF specification [38]. The use of EMOF is

mandatory for the appropriate integration of the verification measures in the *i** framework [135]. Therefore, we have defined the EMOF *i** Metamodel presented in Figure 73. This figure only shows the structural representation of the constructs that are necessary for the application of the proposed verification approach. The complete structure of the defined *i** metamodel is presented in Appendix I.

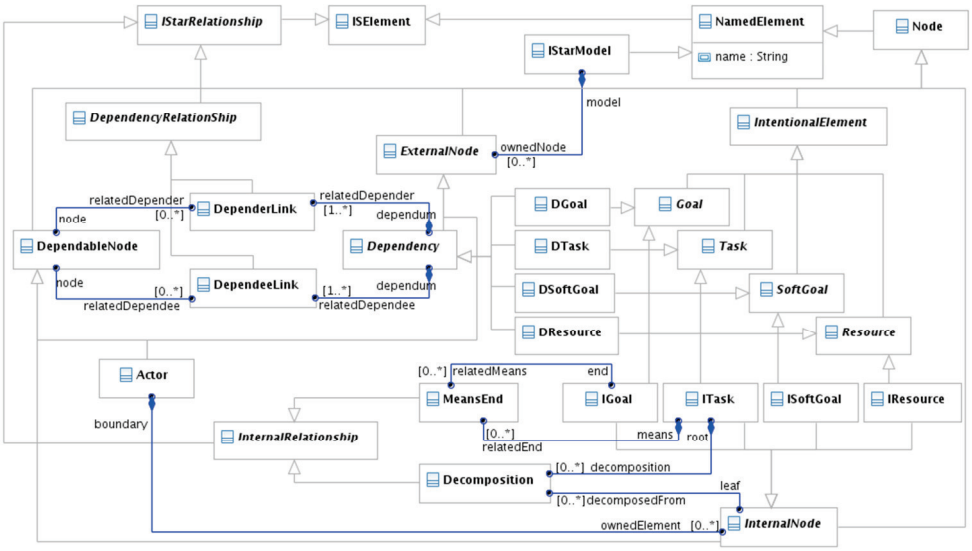

Figure 73. EMOF *i** Metamodel

10.3.3. Step 3: *i** Verification Model Definition

The third step of the process consists in the definition of a verification model. This is an EMOF model that includes the information required for the correct application of the measures (see Figure 74).

The verification model must include those elements that are not present in the reference *i** metamodel, which are also relevant for the correct generation of the corresponding MDD class models according to the transformation guidelines presented in Table 12.

Figure 74 also shows the mapping information that indicates the correspondences among the elements of the verification model and the *i** metamodel.

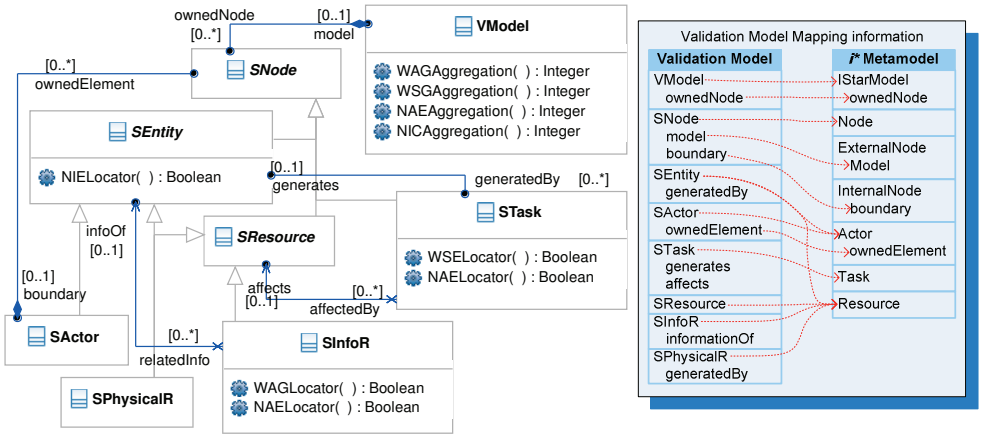

Figure 74. Verification Model and Mapping Information

10.3.4. Step 4: *i** Measures Specification

The fourth step of the process corresponds to the OCL specification of the measures, which must be included in the verification model. This specification is performed by considering the modeling information that is contained in the verification model. Figure 74 shows the names and outputs of the different OCL rules defined. For the measure specification, we have applied the measure patterns presented in [201], specifically, the *aggregation* and *locator* patterns. The *locator* pattern is used to identify the elements involved in the measure evaluation, and the *aggregation* pattern is used to return the final value of the measure. A very useful aspect of the application of these patterns is that the *i** elements that must be fixed can be easily identified by means of the *locator* pattern. For instance, the OCL definition of the measure WAG (Wrong Attribute Generation) is comprised by two OCL rules (see Table 17), these are: the rule *WAGLocator* that identifies the corresponding resources by returning a Boolean value, and the OCL rule *WAGAggregation* that returns the final measure result by aggregating those resources where the OCL rule *WAGLocator* returns true.

In addition, since the proposed measures have been defined for verification purposes, we have introduced a new property in the measure specification, which corresponds to the alert levels that are related to the defined measures. These levels are the following:

1. **Critical:** Indicates that the situation identified by the measure prevents the transformation of the corresponding *i** elements.

2. **Warning:** Indicates that there is a modeling issue that can be fixed to improve the class model generated.

These two alert levels are derived from the separation proposed for the definition of the questions related to the GQM application (see Step 1 of this section). Thus, WAG and WSG measures have a *critical* level, and NAE and NIE measures have a *warning* level.

Table 17. WAG measure specification in the OCL language.

Measure	Subject of Measure	Alert Level
M2: Wrong Attributes Generation (WAG)	*i** Informational Resources	Critical
Context: *VModel*::WAGAggregation() : Integer **Body:** result = self.ownedNode->select(irs|irs.oclIsKindOf(SInfoR)) .oclAsType(SInfoR)->select(irs|irs.WAGLocator())->size() + self.ownedNode->select(act|act.oclIsKindOf(SActor)) .oclAsType(SActor).ownedElement->select(irs| rs.oclIsKindOf(SInfoR)).oclAsType(SInfoR) ->select(irs|irs.WAGLocator())->size()		
Context: S*InfoR*::WAGLocator() : Boolean **Body:** result = self.*infoOf*->isEmpty()		

10.3.5. Step 5: *i** Extensions Generation.

Finally, in the fifth step of the process, the verification model and the OCL specification of the measures are used to generate the metamodel extensions that are necessary to integrate the proposed measures into the *i** framework. These extensions are implemented in a UML profile (see Figure 75), which is generated by means of the proposals presented in [135] and [127]. In [127] is presented an approach for the adaptation of metamodels for generation of UML profiles, and [135] defines a set of transformation rules for automatic UML profile generation. These novel approaches correspond to the Step 2 and Step 3 of the interoperability process proposed (see Chapter IV), which are detailed in Chapter V and Chapter VI. According to the defined interoperability process, these proposals use the mapping information presented in Figure 74.

In general terms, the UML profile generation consists in the generation of one stereotype for each class of the verification model, and the definition of one tagged value for each property (attribute or association end) that has not

correspondence in the target *i** metamodel (non-mapped properties). In particular, for those abstract classes that have the child classes mapped to different classes of the *i** metamodel, only the extensions related to the child classes are represented (stereotype *Actor*). Otherwise, if the abstract and the child classes are mapped to the same class in the *i** metamodel, only the extension of the concrete class is represented (stereotype *SResource*) and the extensions related to the child classes are omitted (stereotypes *SPhysicalR* and *SInfoR*). Additionally, the abstract stereotype *SNode* is not represented since it does not introduces new properties or operations into the *i** metamodel.

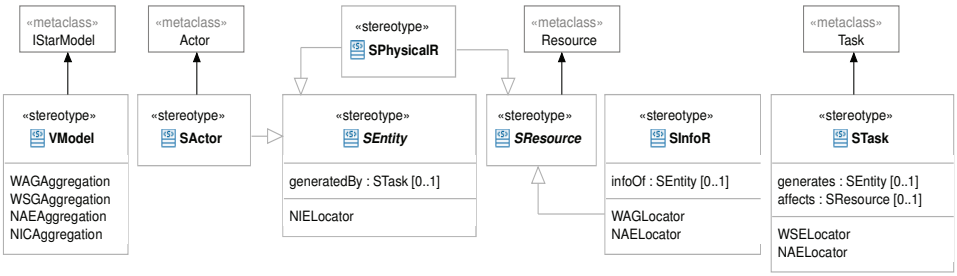

Figure 75. UML Profile to extend the *i** metamodel with the verification measures

The UML profile is a lightweight extension mechanism that does not change the target metamodel, and it has a standardized definition [131] and interchange format [40]. Therefore, it is a suitable alternative for the application of our verification proposal. Other proposals have also considered the use of lightweight extensions for goal-oriented modeling (e.g. [202]).

In the generated UML profile, the elements of the OCL specification must be changed according to the mapped elements of the *i** metamodel (see Figure 74), and the generated stereotypes and tagged values. For instance, the specification for measure WAG (Wrong Attributes Generation) is finally defined as follows:

```
Context: VModel::WAGAggregation() : Integer
Body: result = self.ownedNode->select(irs|irs.isStereotyped(SInfoR))
        .oclAsType(Resource)->select(irs|irs.WAGLocator())->size() +
        self.ownedNode->select(act|act.oclIsKindOf(Actor)).oclAsType(Actor)
        .ownedElement->select(irs|irs.isStereotyped(SInfoR))
        .oclAsType(Resource)->select(irs|irs.WAGLocator())->size()
Context: SInfoR::WAGLocator() : Boolean
Body: result = self.infoOf->isEmpty()
```

It is important to mention that the OCL operation *isStereotype* is not part of the OMG specification and it must be defined or implemented according to the OCL interpreter used. For instance, in ATL this operation can be implemented as follows:

```
helper context UML!Element def : isStereotyped(name:String):Boolean =
not self.getAppliedStereotypes()->select(s|s.name=name)->isEmpty();
```

Finally, for the definition of *i** models extended with the generated UML profile, we have used the eclipse UML2 project. Thus, we take advantage of the already implemented support for UML profiles that this tool provides.

10.4. Applying the *i** Verification Measures

This section exemplifies how the proposed *i** measures are used to verify and improve the generation of the corresponding class model. The process to apply the verification measures is presented in Figure 76.

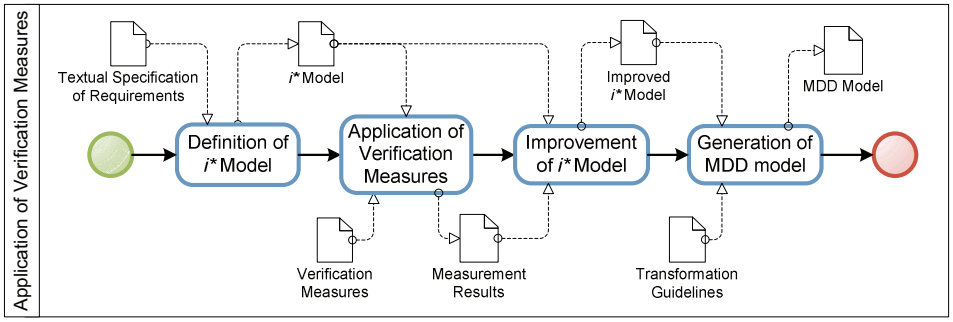

Figure 76. Process for the application of verification measures.

To specify the *i** models, the corresponding EMF editor has been generated by using the *i** metamodel that has been defined as reference (implemented with the Eclipse UML2 tool [126]).

In order to improve the understanding of the *i** models presented in this chapter, the pictures of these models corresponds to manual transcriptions of the defined EMF models using the *i** notation. Therefore, the example *i** model presented in Figure 70 has been extended with the information that is required

for the automatic measures application. Figure 77 shows the *i** model extended with the generated UML profile.

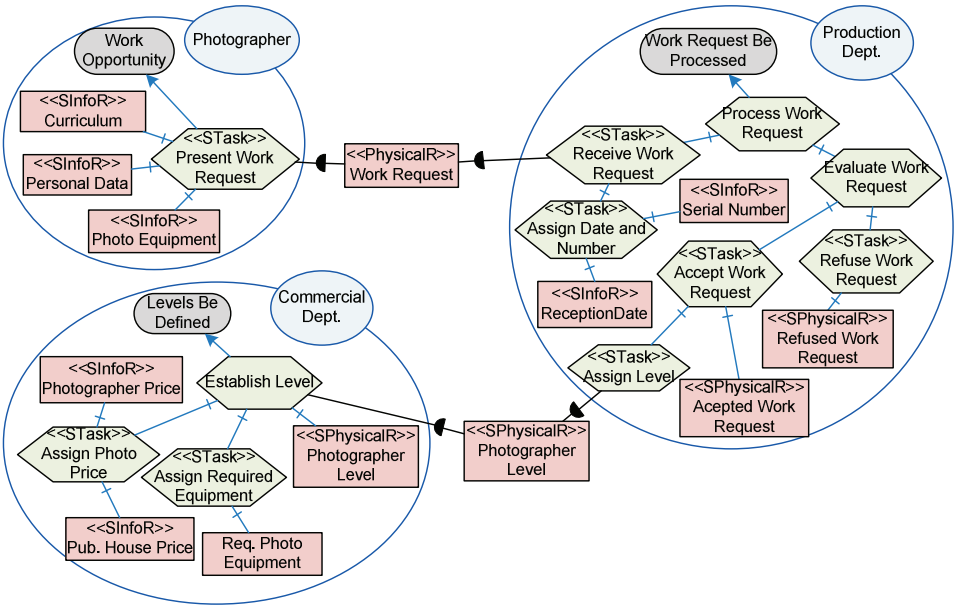

Figure 77. Example *i** Model extended with the generated UML Profile.

Only those *i** elements related to the intended system are considered in the transformation process. These elements are the stereotyped elements.

Table 18 shows the results obtained from the measures evaluation by indicating: 1) the result of the measure (the values obtained from the *aggregation* OCLs); and 2) the *i** elements that return *true* for evaluation of locator OCLs.

Table 18. Results obtained from measures evaluation.

Measure	Alert	Result (Aggregation)	Locator
WAG	Critical	3 Resources	Curriculum, Photo Equipment, Personal Data
WSG	Critical	3 Tasks	Assign Photo Price, Assign Required Equipment, Assign Level
NAE	Warning	15 Elements	All stereotyped informational resources and tasks defined in actors' boundaries (none stereotyped actors in the model)
NIC	Warning	1 Entity	Photographer Level

Table 19 shows the values related to the tagged values of each stereotyped element.

Table 19. Tagged values related to the example *i** Model

TaggedValue	Value	TaggedValue	Value
Curriculum		Photographer Price	
.infoOf	–	.infoOf	Photographer Level
Photo Equipment		Pub. House Price	
.infoOf	–	.infoOf	Photographer Level
PersonalData		Assign Required Equipment	
.infoOf	–	.affects	–
Reception Date		.generates	–
.infoOf	Work Request	Assign Date and Number	
Serial Number		.affects	Work Request
.infoOf	Work Request	.generates	–
Assign Photo Price		Assign Level	
.affects	–	.affects	–
.generates	–	.generates	–
Present Work Request		Refuse Work Request	
.affects	–	.affects	–
.generates	Work Request	.generates	Refused Work Request
Receive Work Request		Accept Work Request	
.affects	–	.affects	–
.generates	Work Request	.generates	Accepted Work Request

Figure 78 shows the class model that may be generated (applying the transformation guidelines presented in Table 12) from the example *i** without considering the information reported by the verification measures.

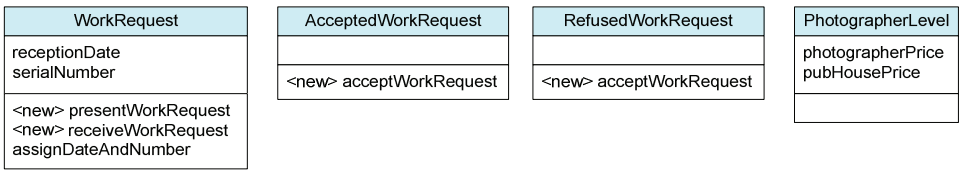

Figure 78. Class model generated from the example *i** model

In Figure 78 can be observed that those elements identified by the critical measures are not present, such as the resource *Curriculum* or the task *Assign Photo Price*. Therefore, it is necessary to improve the defined *i** model in order to assure the transformation of all the selected *i** elements; i.e., to fix the issues related to elements identified by critical measures.

10.4.1. Improving the *i** Models for MDD Model Generation

The results obtained from the measures application provide useful information to fix the detected modeling issues. Thus, it is possible to identify specific fixing guidelines for each measure formulated. For the four measures defined, the alternative guidelines presented in Table 20 have been inferred.

Table 20. Fixing guidelines related to the verification measures

Measure	Wrong Attribute Generation (WAG)
Guidelines	Associate the informational resources to a system entity (stereotyped actor or physical resource).
	Change the kind of the informational resource to *physical resource*.
	Remove the resource from the intended system (un-stereotyped resource).
Measure	Wrong Service Generation (WSG)
Guidelines	Define the owner actor as part of the intended system.
	Indicate if the involved task participates in the generation or affect the state of a system entity (stereotyped actors or physical resources).
Measure	Non-Accessible Element (NAE)
Guidelines	Define the owner actor as part of the intended system.
	Change the informational resource to *physical resource*.
Measure	Non-Instantiable Class (NIC)
Guidelines	Define a new task in the model as production task of the involved entity (stereotyped resource or physical resource).
	Indicate a task that is already defined in the model as production task of the entity (stereotyped resource or physical resource).
	Change the physical resource to *informational resource*.

In addition to the guidelines presented, it is also possible to remove the corresponding element from the intended system (i.e., remove the stereotype), or even remove the element from the *i** model.

However, independently of the guidelines that can be derived from the different verification measures, this information is just a reference for analyst who must decide the guidelines to apply to improve the *i** model. Figure 79 shows the *i** model improved by the analyst after analyzing the results obtained from the application of the verification measures.

In the improved *i** model, the task *Assign Level* affects the state of the new defined actor *Accepted Photographer*. The tasks *Assign Photo Price* and *Assign Photo Equipment* are now related to the resource *Photographer Level*. Another interesting change is the specification of the actor *Req. Photo Equipment* as informational resource. Even though this resource has not been located by the verification measures, the analyst has decided that it must be included in the system as part of the *Photographer Level*.

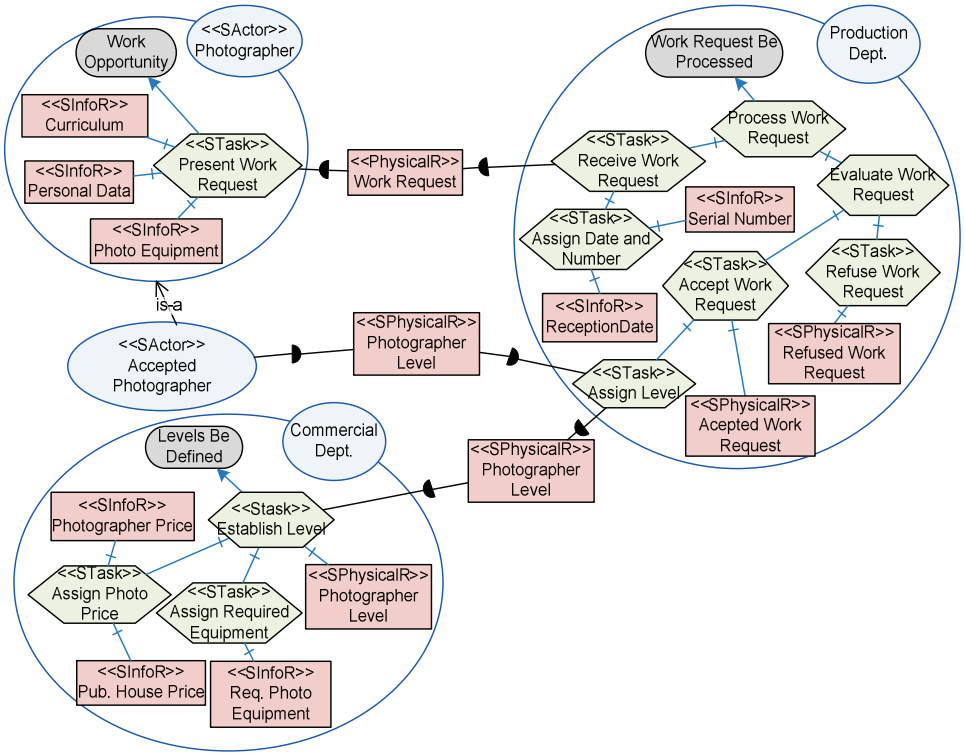

Figure 79. Improved *i** model

The informational resources located by the WAG measure are now defined as information of the actor *Photographer*. The warning related to the NIE measure has been solved by defining the task *Establish Level* as a generation task

for the resource *Photographer Level*. Table 21 shows the tagged values that have been changed in the improved *i** model.

It is important to note that solving the issues identified by the verification measures, the stereotyped elements are properly performed, but also, an improved and detailer requirement representation is obtained. Figure 80 shows the class model generated from the improved *i** model.

Table 21. Tagged values changed in the improved *i** Model

TaggedValue	Value	TaggedValue	Value
Curriculum		Assign Required Equipment	
.infoOf	Photographer	.affects	Photographer Level
Photo Equipment		.generates	
.infoOf	Photographer	Assign Level	
PersonalData		.affects	Accepted Photographer
.infoOf	Photographer	.generates	–
Req. Photo Equipment		Establish Level	
.infoOf	Photographer Level	.affects	–
Assign Photo Price		.generates	Photographer Level
.affects	Photographer Level		
.generates	–		

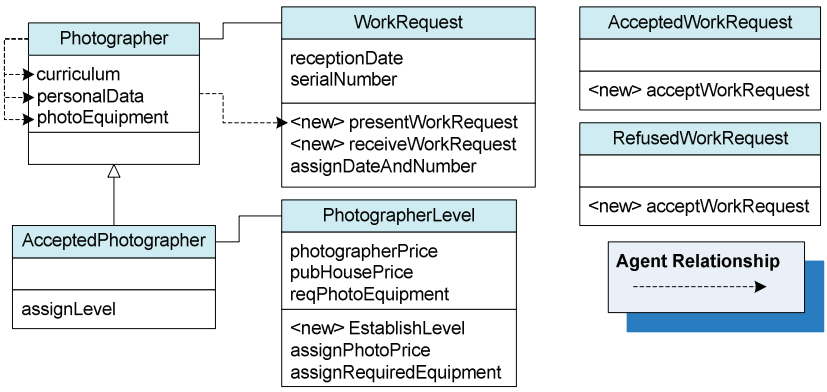

Figure 80. Class model generated from the improved *i** model

Figure 80 shows that the class model generated from the improved *i** model has a more detailed system specification. Essential elements generated from the

improved *i** model are the classes *Photographer* and *AcceptedPhotographer*. Also, associations among classes have been generated. In summary, all the stereotyped elements of the *i** model have been transformed to conceptual constructs of the target class model. Thus, the MDD model represents all the system requirements considered.

Since the generated class model is an initial MDD model, it must be refined at design time. Some possible refinements are the specification of the specializations that exist between the class *PhotoWorkRequest* and the classes *AcceptedWorkRequest* and *RefusedWorkRequest*. Also, the cardinality of the associations and the appropriate specification of the services must be defined.

In the improved *i** model, there still exist warning issues related to the NAE measure. These warning issues are derived from the stereotyped resources and tasks that are defined inside of the boundaries of the actors *Production Dept.* and *Commercial Dept.*, which are not considered as part of the system-to-be by the analyst. In this case, the analyst has considered that an administrator user must be specified at design time to visualize and execute the corresponding resources and tasks. However, since the NAE measure is just a warning measure, it does not prevent the correct transformation of the *i** model. It just indicates that the MDD constructs obtained from identified elements must be refined at design time to obtain a complete specification.

10.5. Evaluating the Verification Approach

To evaluate the verification approach, we have focused on the efficacy of the measures obtained with the proposed definition process to achieve the completeness of the generated MDD model.

The ISO 9126 standard [203] distinguishes between two kinds of completeness: 1) the completeness of a system with respect to the requirement specification; and 2) the completeness of the functionality that a system must supports. Thus, the first kind of completeness is related to the completeness of the MDD models in relation to the system requirements that are defined in the *i** models. The second kind of completeness is related to the completeness of the initial MDD model regarding to the functionality of the software system; i.e., the completeness of the MDD model to perform the automatic model compilation.

The evaluation of our verification approach has been conducted as a laboratory experiment, which has been designed using the framework proposed by Wohlin et al. [204] for Empirical Software Engineering. The research question addressed by the experiment is stated as:

RQ1: *Is the completeness of the generated MDD model supported by the measures obtained from the application of the proposed verification approach?*

The rest of this section provides details of the design of the laboratory experiment, as well as the results obtained from the experiment execution.

10.5.1. Subjects, Variables, and Hypothesis

Four subjects were selected to participate in the study: two *i** analysts (identified as ANA1 and ANA2) and two measurement experts (identified as EXP1 and EXP2). These subjects are Computer Science PhD students from the Universidad Politécnica de Valencia, which have similar backgrounds in the *i** framework and the OO-Method MDD approach.

The independent variables in the experiment correspond to the Photography Agency *i** models, which have been defined by the *i** analysts. The first *i** model (called ISTAR1) is already detailed in Section 10.4 (see Figure 77). The second defined *i** model (called ISTAR2) is presented in Figure 81.

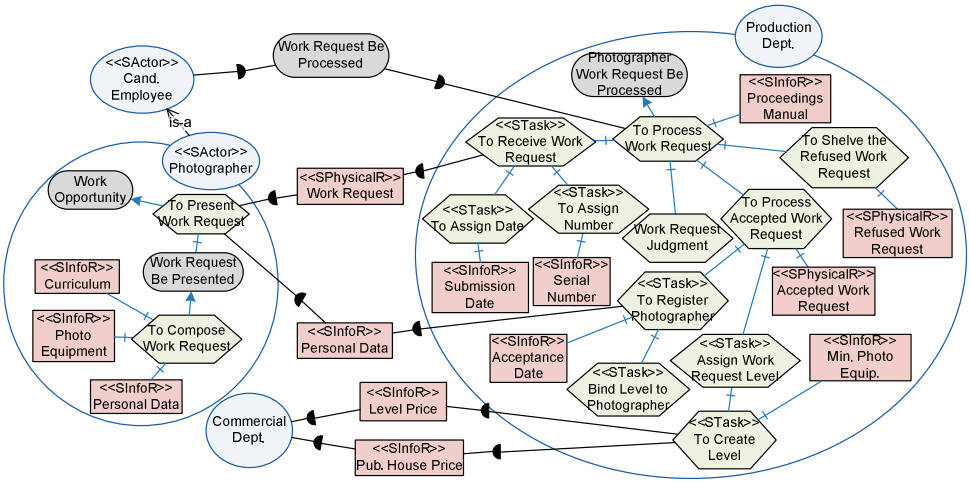

Figure 81. Second *i** Model (ISTAR2) for Photography Agency Description

Table 22 presents the information related to the tagged values of ISTAR2 and Figure 82 presents the initial MDD model (MODEL2) generated from ISTAR2 without the information of the verification measures.

Table 22. Tagged values related to the Second *i** Model for Photography Agency

TaggedValue	Value	TaggedValue	Value
Curriculum		To Create Level	
.infoOf	Photographer	.affects	–
Photo Equipment		.generates	–
.infoOf	Photographer	To Receive Work Request	
PersonalData		.affects	–
.infoOf	Cand. Employee	.generates	Work Request
Level Price		To Assign Date	
.infoOf	–	.affects	Work Request
Proceedings Manual		.generates	–
.infoOf	–	To Register Photographer	
Min. Photo Equip.		.affects	Photographer
.infoOf	–	.generates	Photographer
Acceptance Date		Bind Level to Photographer	
.infoOf	–	.affects	Photographer
Submission Date		.generates	–
.infoOf	Work Request	To Assign Number	
Serial Number		.affects	Work Request
.infoOf	Work Request	.generates	–
Pub. House Price		Assign Work Request Level	
.infoOf	–	.affects	Accepted Work Request
		.generates	–

The quantitative dependent variables considered in the experiment are the following:

1. Number of informational resources that cannot generate the corresponding class attributes in the initial MDD model. Obtained from WAG measure.

2. Number of tasks that cannot generate the corresponding service definitions in the initial MDD model. Obtained from WSG measure.
3. Number of tasks and informational resource that generate non-accessible elements in the initial MDD model. Obtained from NAE measure.
4. Number of actors and physical resources that generate non-instantiable classes in the initial MDD model. Obtained from NIC measure.

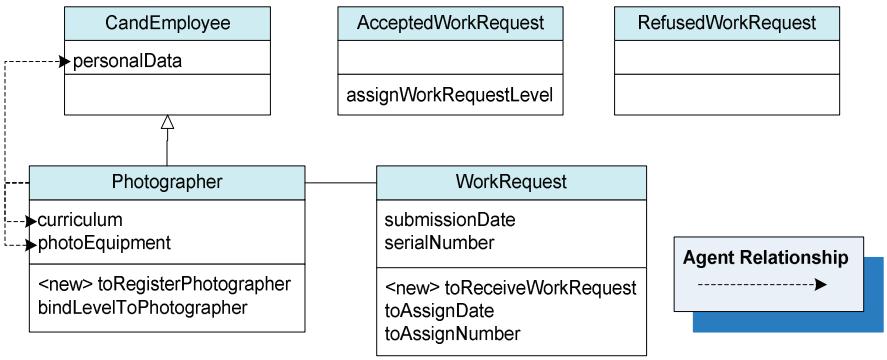

Figure 82. Initial class model generated from ISTAR2 without improvements

The following hypotheses related to the critical measures and the warning measures are considered to answer our research question:

H_{RCOM}: The critical measures allow the verification of all the system requirements that are defined in the extended *i** model to generate the corresponding MDD conceptual constructs.

H_{CCOM}: The warning measures allow the verification of those *i** elements that can be improved to generate a more complete specification of the initial MDD model, which represents the functionality of the final software product.

To test H_{RCOM}, each *i** element related to the intended system (the stereotyped elements in the extended *i** model) must have a direct relation with the constructs generated in the initial MDD model (the OO-Method class model in the experiment).

To test H_{CCOM}, the improvements performed to the *i** model with the information obtained from the warning verification measures must generate a more detailed specification of the initial MDD model.

10.5.2. Instruments and Experimental Tasks

Figure 83 shows the tasks performed in the experiment, which are modeled with the BPMN notation [205]. These tasks are described below.

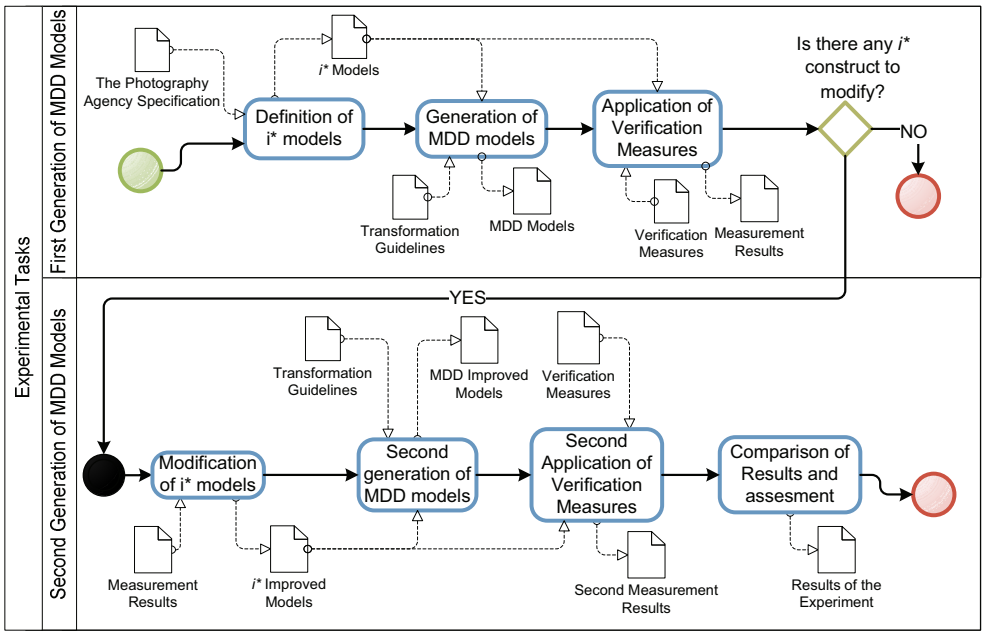

Figure 83. Experimental tasks

Task 1. ***Definition of i* models.*** Each one of the two involved *i** models (see Figure 77 and Figure 81) is defined by one *i** analyst, which considers the organizational description presented Section 10.2.3. The defined *i** models include the information that is required for the automatic application of the transformation guidelines, as described in Section 10.2.4.

Task 2. ***Generation of MDD models.*** Each measurement expert performs a class model generation from one of the defined *i** models by applying an ATL transformation script. The verification measures are not applied in this task.

Task 3. ***Application of verification measures.*** Each measurement expert applies the verification measures over its corresponding *i** model.

Task 4. ***Modification of i* models.*** The analysts use the results obtained from Task 3 to improve their corresponding *i** models.

Task 5. ***Second generation of MDD models.*** The measurement experts generate a new class model from the improved version of the *i** model that they have transformed in Task 2.

Task 6. ***Second Application of verification measures.*** The measurement experts apply the verification measures over the improved *i** model.

Task 7. ***Comparison of results and assessment.*** The results obtained from Tasks 2, 3, 5, and 6 are compared and analyzed by the two measurement experts to check the hypotheses proposed.

In addition to these sequential tasks, the measurement experts have controlled all the experiment execution.

Regarding the instruments, in addition to the models themselves, the instruments used in the experiment were: the Eclipse Model Development Tools [134], the EMF editor for the *i** metamodel extended with the UML profile related to the verification measures and transformation extensions (see Figure 75), the ATL scripts to transform the *i** models to MDD models according to the transformation guidelines presented in Table 12, and tables filled according to a predefined template to keep the results of the experiment.

10.5.3. Execution and Data Collection Procedures

To perform the experiment, the *i** analysts were located in separated rooms to avoid any kind of influence on each other's results. The *i** models were defined manually to prevent that the use of the EMF editor (that does not provide *i** notation) affects to the appropriate analysis of the business. In tasks 2 and 5, the measurement experts translate the hand-made *i** models with the corresponding EMF representation in order to apply the verification measures and to generate the corresponding MDD models automatically. No time limit was set for any experimental tasks, such as the EMF models specification, the generation of the corresponding MDD models, the application of the verification measures, the improvement of the *i** models, etc.

In this study, data triangulation was considered (which refers to using more than one data source or collecting the same data on different occasions) since we used two sources (the two *i** models). Thus, the following common steps were defined to collect data from the two *i** models in the study:

1. Each *i** model was transformed into an OO-Method class model by measurement experts applying an automatic transformation process.
2. Work diaries were completed by each measurement expert for each model transformed. In those diaries, the information obtained from the verification measures was registered.

10.5.4. Results: Analysis and Interpretation Issues

In the first generation of the MDD models, the resultant models suffered from several defects related to their completeness. Table 23 shows the amount of constructs of the *i** models that must be transformed, which correspond to the stereotyped elements. Also, Table 23 shows the amount of effectively transformed elements. Thus, it is clear that not all the stereotyped elements were transformed into constructs of the class models, i.e.; MDD models are not complete regarding to the requirements.

Table 23. First generation of the MDD models.

*i** Model	Stereotyped Elements	Transformed Elements	MDD Model	MDD Elements
ISTAR1	19	13	MODEL1	4 classes, 4 attributes, 5 services. (Total = 13)
ISTAR2	23	17	MODEL2	5 classes, 6 attributes, 6 services, 1 association, 3 agent relationships, 1 generalization. (Total = 22)

Then, EXP1 and EXP2 apply the verification measures to the defined *i** models. The results obtained are presented in Table 24, which shows for each *i** model the measures applied, the alert level of the measures, the result of the measure obtained from the evaluation of *aggregation* OCLs, and the *i** elements that return *true* from the evaluation of *locator* OCLs.

It is important to point that the verification measures can be used to define additional measures to obtain relevant information of the defined *i** models. For instance, we have defined the following measure to obtain information about of the completeness of the MDD model to be generated.

$$PTE = \left(\frac{TSE - (WAG + WSG)}{TSE}\right) \times 100$$

The measure PTE (*Percentage of Transformable Elements*) obtains the percentage of *i** elements related to the intended system that are transformed in elements of the target MDD model. PTE is calculated using TSE, WAG, and WSG measures.

Table 24. Application of the verification measures to ISTAR1 and ISTAR2

Model	Measure	Alert Level	Measurement Result	Locator
ISTAR1	WAG	Critical	3 Resources	Curriculum, Photo Equipment, Personal Data
	WSG	Critical	3 Tasks	Assign Photo Price, Assign Required Equipment, Assign Level
	NAE	Warning	15 Elements	All stereotyped informational resources and tasks defined in actors' boundaries (none stereotyped actors in the model)
	NIC	Warning	1 Entity	Photographer Level
ISTAR2	WAG	Critical	5 Resources	Level Price, Proceedings Manual, Min. Photo Equip., Acceptance Date, Pub. House Price
	WSG	Critical	1 Task	To Create Level
	NAE	Warning	12 Nodes	All the stereotyped informational resources and tasks defined in the Production Dept. Boundary
	NIC	Warning	3 Entities	Cand. Employee, Accepted Work Request, Refused Work Request

TSE (*Total Stereotyped Elements*) counts the elements identified to be part of the intended system (the stereotyped elements). For ISTAR1, TSE = 19 and, for ISTAR2, TSE = 23 (see Table 23).

WAG and WSG correspond to the critical verification measures. For ISTAR1, WAG = 3 and WSG = 3. For ISTAR 2, WAG = 5 and WSG = 1 (see Table 24). Note that the sum of the critical measures (WAG and WSG) is coincident with the difference of transformed *i** elements (see Table 23).

Thus, for ISTAR1 we obtain PTE = 68,4 %. It means that only the 68,4% of the *i** stereotyped elements can be transformed in the MDD model generation.

i.e. 31,6% of elements related to the system requirements will not be represented in the software model. For ISTAR2, we obtain PTE = 73,9 %.

The warning measures support the identification of those *i** elements that can be improved to obtain a more complete specification of the initial MDD model generated. Thus, we have defined the following measure to identify this situation in the improved *i** models:

$$WIP = \left(\frac{(IMDD - ((OMDD + WAG + WSG))}{OMDD} \right) \times 100$$

The measure *Warning Improvement Percentage* (WIP) returns the percentage of new MDD constructs of the improved MDD models obtained with the information of the warning measures. WIP is calculated using IMDD, OMDD, WAG, and WSG.

IMDD (Improved MDD) corresponds to the number of MDD constructs generated from the Improved *i** model. OMDD (Original MDD) corresponds to the number of MDD constructs generated from the original *i** model. WAG and WSG correspond to the critical measures previously defined. In the experiment, the WIP measure is evaluated after the second generation of MDD models.

The next task in the experiment (Task 4) corresponds to improve the *i** models using the information obtained from the verification measures WAG, WSG, NAE, and NIE. Following, we enumerate the improvements performed by ANA1 to the model ISTAR1 (see Section 5.1):

1. 1 actor, 1 is-a relationship, and 1 task were added to the *i** model.
2. 2 actors were added to the system requirements.
3. 4 resources and 4 tasks were modified in the *i** model.

The same cognitive process explained in Section 10.4.1 to improve the ISTAR1 model is applied by ANA2 to obtain the improved ISTAR2 model (see Figure 84). The improvement actions performed were the following:

1. 2 actors were added to the system requirements
2. 1 goal, and 1 resource were added to the *i** model.
3. 2 tasks and 6 resources were modified in the *i** model.
4. 1 stereotype application was changed to physical resource in the *i** model.

An interesting benefit that emerged while fixing the elements of ISTAR2 that are identified by the critical measures was that the analyst ANA2 detected a

mistake in the understanding of the organizational description. The analyst initially defined the actor *Production Department* as responsible for the levels definition. However, the actual responsible is *Commercial Department*. As a consequence, the analyst defined a new physical resource *Level*, where the task *To Create Level* is the production task for this physical resource. Thus, resources *Price Min.*, *Photo Equip.*, and *Pub. House Price* are defined as informational resources of the *Level* resource. Furthermore, in contrast to the reasoning performed by the first analyst (ANA1), the second analyst (ANA2) considered that all the actors involved in the *i** model must be part of the system-to-be. Thus, the improved *i** model did not generate non-accessible elements in the MDD model (measure NAE = 0). Additionally, the resource *Proceeding manual* is changed from informational entity to physical entity. Figure 84 shows the improved ISTAR2 model, Table 25 the tagged values changed, and Figure 85 shows the MDD model generated.

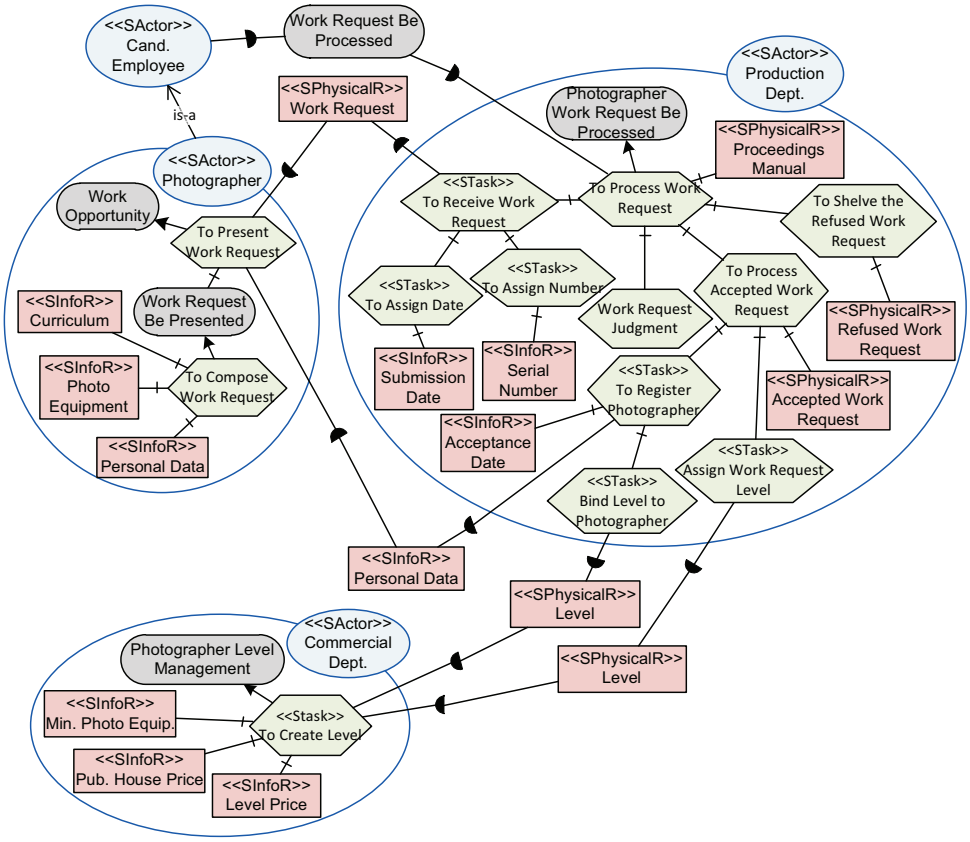

Figure 84. Second *i** model (ISTAR2) improved with the verification measure results

Table 25. Tagged values changed in the improved *i** Model

TaggedValue	Value	TaggedValue	Value
Level Price		Acceptance Date	
.infoOf	Level	.infoOf	Photographer
Min. Photo Equip.		To Create Level	
.infoOf	Level	.affects	–
Pub. House Price		.generates	Level
.infoOf	Level		

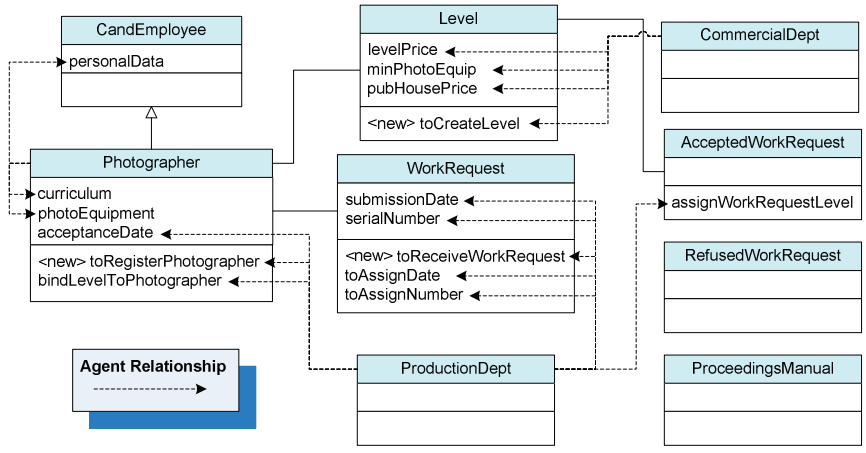

Figure 85. Class model obtained from the improved version of the second *i** model

Table 26 shows the results obtained from the transformation of the improved versions of the models ISTAR1 and ISTAR2.

Table 26. Second generation of the MDD models.

Improved *i** Model	Stereotyped Elements	Transformed Elements	Improved MDD Model	MDD Elements
ISTAR1	23	23	MODEL1	6 classes, 8 attributes, 9 services, 2 associations, 4 agent rel., 1 generalization (Total=30)
ISTAR2	25	25	MODEL2	9 classes, 9 attributes, 7 services 3 association, 16 agent rel., 1 generalization (Total=45)

Table 27 shows the results obtained from the application of the verification measures to the improved *i** models. For ISTAR1, the measures WAG, WSG, and NIC are equal to 0, which means that the improved model ISTAR1 generates correctly attributes, services, and instantiable entities. Only NAE was greater to zero (NAE=16), which means that 16 elements of the generated MDD model (MODEL1) do not have agent relationships defined.

For ISTAR2, the measures WAG, WSG and NAE are equal to 0, which means that that the improved model ISTAR2 generates attributes, services, and accessible elements correctly. Only NIC was greater to zero (NIC=6). In fact, NIC's value is even greater than the result obtained from the initial ISTAR2 model (NIC=3). This situation is produced by the two new actors defined as part of the system, and the change in the stereotype application of the resource *Proceeding Manual*, which is defined now as a physical resource. However, this warning measure does not prevent the proper generation of the MDD model (MODEL2).

Table 27. Experiment results

Measures →	WAG	WSG	NAE	NIC	PTE	WIP
First Generation (Initial *i** Models)						
ISTAR1	3	3	18	1	68,4%	–
ISTAR2	5	1	13	3	73,9%	–
Second Generation (Improved *i** Models)						
ISTAR1	0	0	16	0	100%	84,6%
ISTAR2	0	0	0	6	100%	77,3%

With the results obtained in the experiment we can test the hypotheses H_{RCOM} and H_{CCOM}, and consequently answer our research question.

The experiment shows that by fixing the issues identified from the application of the critical measures (WAG and WSG) in the improved *i** models ISTAR1 and ISTAR2, the completeness of the resultant MDD models (improved MODEL1 and MODEL2) is achieved according to the system requirements. In both *i** models, 100% of the stereotyped elements are transformed into the corresponding MDD constructs (see PTE measure in Table 27). Therefore, we can state that the Hypothesis H_{RCOM} is demonstrated.

Also, the experiment results shows that fixing the issues identified by the warning measures (NAE and NIC), the completeness of the MDD models in

relation to the system functionality is higher. This is observed in the amount of MDD constructs generated from the improved *i** models in relation to the original *i** models (see WIP measure in Table 27). For ISTAR1, we obtain that the improvements performed from warning measures generate 84,6% of additional MDD constructs. For ISTAR2, we obtain that the improvements related to warning measures increase the number of generated MDD constructs in 77,3%. Thus, since MDD constructs are directly representing functionality of the final software system (such as system users), the hypothesis H_{CCOM} is also demonstrated with the results obtained.

10.6. Overall Analysis

A first element to comment is the relevance of using a measure definition process as starting point of our verification approach instead of a direct and intuitive definition of OCL verification rules. This decision comes from the maturity that the measurement specification has in the software engineering context, where we can found sound frameworks for the definition and implementation of measures. This provides a more systematic and rigorous schema for appropriate identification of properties that must be measured and, in the context of this chapter, verified. In addition, we can take advantage of existing measures that are already defined in the context of *i** and object-oriented modeling.

Another relevant point to comment is related to the benefits that the approach proposed for the integration of the verification measures in the *i** framework provides. One of the main advantages of this integration approach is that the entire measure specification is performed by following the model-driven philosophy, where the measures and the required modeling information are specified in a verification model by using current metamodeling standards. The extensions over the *i** framework are defined by means of a UML profile that does not alter the original *i** metamodel specification, which permits the compatibility with existing technologies that use the same metamodel as reference. We have considered mechanisms to automate the generation of this UML profile. With this, the main effort in the application of the verification proposal is in the appropriate definition of the required measures, the correct specification of the involved properties and OCL rules in the verification model, and the definition of the mapping between the verification model and the target *i** metamodel. Thus, once the verification framework is defined for a particular

MDD method (like OO-method in this example scenario); it can be used with no or little modification over and over in different projects.

Finally, it is important to mention that for the application of our verification proposal, we did not find tools that provided transparent support for all the modeling features considered, and hence, additional programming effort was necessary. However, the current development of tools that provide support to the standards considered (such as the Eclipse UML2 project [8]) and the increasing number of research works that take advantage of these technologies are indictors that improved tools will appear in a not-too-distant future. This also motivates the emergence of new approaches for integration and verification to improve the MDD capabilities and the quality of the generated software products.

10.7. Conclusions

The interoperability of *i** models for requirements elicitation in MDD process is a step beyond going from Model-Driven Development (MDD) to Model-Driven Engineering (MDE) [13], where different modeling approaches can interoperate to obtain improved software products [32]. This chapter has presented new results in this direction by presenting a process for the definition and integration of measures that guarantee the completeness of the MDD interoperability. The main advantages of the proposed verification approach are the following:

1. The definition of the verification measures is driven by a systematic process that helps in the correct identification of the elements that must be verified.
2. The evaluation of the verification measures is automatic, which implies a time and effort reduction with respect to manual verifications.
3. Specific alert levels are defined to distinguish the constructs that must be fixed from the constructs that can be improved.
4. The definition and implementation of the verification measures is performed by using current technologies and standards.

It is important to remark that, although our validation framework goes one step beyond in the MDD process by considering goal-oriented models as starting point, we are not aiming at getting a fully automated transition from these types of models into traditional conceptual models. Even though the evaluation of the OCL rules related to the verification measures is automated, and the kind of

measure provides information about the problem and possible solutions, the analyst is still responsible of determining the changes that are necessary to fix *critical issues*, and to decide whether a change (model improvement) is necessary in the case of *warning issues*.

The verification measures presented in this chapter are a subset of the measures that we have defined for the application of *i** models into the OO-Method MDD process. The selected verification measures are presented with the intention of clarifying and exemplifying the verification approach proposed. These verification measures have been applied to OO-Method development scenarios and validated by experts from requirement engineering and MDD areas.

Finally, it is important to remark the results obtained from the application of the verification approach no only guarantee the interoperability of the *i** models into MDD process, but also, it validates the applicability of the MDD interoperability approach presented in this thesis by means of empirical results.

Chapter XI.
Conclusions

In the present software development context, where the model-driven paradigm is the clear trend for current and future software development solutions, it is possible to find several modeling approaches that are related to different application domains.

Upon further examination of the specification of the modeling approaches, equivalences among the defined conceptual constructs can be observed. The number of equivalences should increase for those modeling approaches related to a same application domain (such as requirement analysis, business processes, software design, etc.). In this thesis, we have used these equivalences as integration points to interchange modeling information among different modeling approaches, thus achieving the interoperability in a MDD process. Novel proposals have been defined to properly support this MDD interoperability, thereby dealing with specific interoperability issues that may prevent an appropriate interchange of modeling information.

This chapter summarizes the contributions obtained from the work performed in this thesis. Future research lines and the publications generated during this thesis are also detailed.

The work presented in this thesis clearly demonstrates that it is possible to achieve MDD interoperability by applying an appropriate process, which generates the necessary interoperability artifacts to integrate and interchange modeling information. Moreover, current model-based standards and technologies can be used to provide suitable support to the proposed MDD interoperability with minor adaptations to the implemented tools.

In the application and verification of our proposal, we show the interoperability of different modeling approaches that are related to the same abstraction level. In particular, we apply the UML and OO-Method modeling languages to define a suitable model specification, which can be compiled in the final software product by means of the industrial tools implemented for the OO-Method approach.

The interoperability among modeling solutions related to different abstraction levels is also presented, where *i** analysis models are used as a starting point to automatically generate the design models related to the OO-Method MDD approach. Thus, the requirement models are used as active artifacts in the development process, and not just as reference documentation. With this multi-level interoperability schema, the requirements, the design, and the implementation stages of a MDD process are covered.

A model-driven proposal to verify the correct MDD interoperability is also presented. This proposal is applied to assure the completeness of the interoperability related to *i** requirement models in the OO-Method MDD process. The verification proposal has been empirically verified obtaining successful results, not only for MDD interoperability but also for the improvement of the requirement models involved and the MDD models generated. The definition of this approach to verify the MDD interoperability is oriented to support real development contexts, where the defined models may present defects that prevent correct model transformations or compilation.

Even though we have obtained good results from the application of our interoperability approach in the interoperability scenarios presented, we not consider our approach to be the definitive solution for MDD interoperability. Additional work and further research are needed to achieve this ambitious objective. However, this thesis makes an important effort in this direction, which can be used as a starting point to obtain a model-driven development process closer to real model-driven engineering, which takes advantage of different modeling approaches, standards, tools, and knowledge generated from different research areas to improve the MDD processes.

The rest of this chapter is organized as follows: Section 11.1 indicates the main contributions obtained in this thesis. Section 11.2 presents future research lines. Finally, Section 11.3 details the publications generated from the work presented in this thesis.

11.1. Thesis Contributions

In general terms, the contributions obtained from the work in this thesis are directly related to the goals initially stated, and, hence, the main contribution is the *approach for the interoperability of different modeling languages in a common MDD process*. There are also additional contributions that are oriented to achieving the specific objectives that support this main goal. Following, these additional contributions are presented by indicating how they support the specific objectives stated in the first chapter of this thesis.

C1. An MDD interoperability Model

This model states the elements that are necessary to achieve MDD interoperability. According to the defined MDD interoperability model, MDD interoperability is defined in terms of Technical, Semantic, and Syntactic interoperability. Also, the proposed model indicates that MDD interoperability can be automated if an appropriate procedure is used to coordinate the applications, infrastructure, and data that are related to the different modeling artifacts. The proposed MDD interoperability model is presented in Chapter IV. An additional contribution has been obtained from the MDD interoperability model, which corresponds to the following:

1. **A MDD interoperability process.** This process has been defined together with a set of application guidelines to obtain a sound interoperability procedure.

C2. The Integration Metamodel Proposal

This particular metamodel is defined to indicate the semantic equivalences among the modeling languages to be integrated. These equivalences are specified by means of metamodel mappings. Thus, the Integration Metamodel is used as a pivot metamodel that fixes those mapping issues that may prevent an automatic

interchange of models. It also provides a suitable structure for the automatic generation of metamodel extensions to integrate the modeling languages involved, which is the key to prevent the loss of modeling information during the interchange of models. The Integration Metamodel is presented in Chapter V. For the specification of an appropriate Integration Metamodel, two additional contributions have been generated. These contributions are the following:

1. **A systematic approach for Integration Metamodel definition**. This systematic approach is based on an iterative process that guides the fixing of integration problems and the specification of appropriate metamodels mappings. Thus, a correct Integration Metamodel is obtained.
2. **Verification rules to assure the modeling language integration.** These rules automatically identify the integration issues that exist between two modeling languages by means of the analysis of the defined metamodel mappings.

C3. An approach for automatic UML profile generation

This approach uses an Integration Metamodel as input to generate the lightweight metamodel extensions. These extensions are required to customize the abstract syntax of a target modeling language with the modeling information of the involved MDD approach so that the interchange of modeling information between the customized modeling language and the MDD tools can be properly performed. The proposal for the UML profile generation also generates the necessary mappings to perform the model interchange by means of automatic model transformations. The UML profile generation approach is presented in Chapter VI.

C4. A proposal for automatic interchange of models.

During the definition of the Integration Metamodel and the generation of the metamodel extensions, mapping information among the metamodels of involved modeling languages and the metamodel of the target MDD approach is obtained. Thus, the interchange proposal uses this mapping information as input for the generation of model transformation rules, which can be implemented with current model-to-model transformation technologies (such as ATL or QVT). In other words, the generated transformation rules allow the modeling information to be automatically interchanged, which supports the automatic interoperability of the customized modeling language and the MDD process. This interchange proposal is presented in Chapter VII.

C5. An interoperability framework for UML and MDD processes.

A specific interoperability framework has been obtained from the interoperability scenario proposed for UML and OO-Method. This framework can be used as reference for the interoperability of modeling languages that represent different views of a model at the same abstraction level. Specifically, the level considered in the interoperability scenario related to UML and OO-Method is the design level. This interoperability framework is presented in Chapter VIII. Furthermore, the development of this interoperability framework has produced additional contributions, which correspond to the following:

1. **Customization of the UML association for its application into MDD processes.** This customization has been performed by means of a UML profile, which is automatically generated according to the application of the MDD interoperability process proposed [206]. The generated UML profile contains the extensions that are necessary to integrate the abstract syntax that supports the OO-Method semantics into UML.
2. **Definition of an EMOF metamodel for the OO-Method approach.** The industrial OO-Method implementation is based on an implicit metamodel specification that is associated to the OO-Method model compiler. However, an explicit OO-Method metamodel that is based on the current metamodel standards has not yet been defined. Therefore, the OO-Method metamodel is a relevant contribution that is generated from the interoperability scenarios [32, 206]. This metamodel can be used as reference for other MDD approaches and future research associated to the OO-Method approach.
3. **Implementation of industrial tools for interchange of UML and OO-Method models.** In this thesis, two model transformation tools [132, 135] have been implemented to support the exportation of OO-Method models to UML-based tools and the importation of UML models into the OO-Method modeling suite. These tools are currently part of the industrial suite for management of OO-Method models, which is called Olivanova [64].

C6. An interoperability framework for Requirement Modeling and MDD

This framework is related to the scenario developed for the linking of *i** models with the OO-Method development process. It can be used as a reference for the interoperability of modeling approaches that are related to different abstraction levels. Specifically, the levels considered in the *i** and OO-Method interoperability scenario are the requirement and design levels. This

interoperability framework is presented in Chapter IX. Additional contributions have been obtained from the *i** and OO-Method interoperability. These contributions are the following:

1. **Definition of *i** extensions and transformation rules for the generation of MDD-oriented models from *i** models.** The *i** extensions and transformation rules provide suitable model-based support for the automatic interoperability of goal-oriented requirement models and MDD-oriented models. These elements can be used as a reference for MDD approaches that want to integrate requirement models into their development processes.
2. **Definition of an *i** EMOF Metamodel.** In the *i** context, there is not a standardized *i** metamodel, and, in general terms, the existing metamodel proposals are not EMOF compliant. Therefore, the EMOF metamodel that is implemented in the context of the *i** and OO-Method interoperability scenario can be used as a reference for other MDD approaches. Specifically, to facilitate the implementation of open-source tools based on the eclipse development framework.

C7. An approach for verification of MDD interoperability

At the moment of applying the transformation rules that have been generated from the application of the MDD interoperability process, we have detected that there may be certain modeling issues in the defined models that prevent the correct execution of the model transformations. Hence, a specific approach has been defined to verify these interoperability issues. This verification approach is based on the artifacts obtained from the application of the interoperability process. It assures the correct interchange of model information through specific verification measures, which are integrated into the customized modeling languages for their automatic evaluation. This interoperability verification approach is presented in Chapter X. Additional contributions have been obtained from the development of this verification approach. These contributions are the following:

1. **A systematic approach for the generation of automatic verification measures.** This systematic approach has been defined to assure the correct definition of verification measures, which guarantee the completeness of the transformations that are involved in a MDD interoperability framework. This systematic approach involves the definition of a verification model, which allows the automatic integration of the verification measures into the modeling languages by means of UML profile generation.

2. **A set of verification measures for interoperability of *i** and MDD**. These verification measures have been defined for the *i** and OO-method interoperability scenario. We have focused on the definition of a set of verification measures that can be generalized to other object-oriented MDD processes.

Finally, since models are the main resource of MDD processes, the MDD interoperability approach proposed in this thesis could also achieve the interoperability among different MDD processes by using the models as linking point. Therefore, it is possible to take advantage of existing model-oriented technologies and to facilitate the reuse of knowledge and experience across the MDD community. The next section shows the future works that we have planned to develop in this direction together with new proposals for the improvement of MDD interoperability.

11.2. Future work

As future work, we plan to continue our research by focusing on three main lines: the improvement of the MDD interoperability approach, the construction of MDD tools that support the proposed interoperability model, and the application of the interoperability proposal to new MDD interoperability scenarios. We have started to address these three research lines in the following works.

Improvement of the MDD interoperability approach.

1. **Patterns for Integration Metamodel definition.** We are currently working on a set of patterns to provide semi-automatic support to the generation of an Integration Metamodel. These patterns solve the integration conflicts identified by means of the rules presented in Chapter V.
2. **Definition of a new lightweight customization mechanism.** Even though UML profiles provide suitable features for the application of the proposed MDD interoperability approach, there are certain issues related to this extension mechanism that must be fixed in order to obtain an improved interoperability solution. Thus, we plan to define an improved lightweight customization mechanism for MDD interoperability.

3. **Empirical verification and validation.** We are preparing new empirical experiments to verify and validate our interoperability proposal. Specifically, we are currently planning a case study in the context of the *i** and OO-Method approach. Furthermore, we plan to develop additional experimentation related to new MDD interoperability scenarios.

Tools to support the MDD interoperability.

1. **A tool for automatic application of the interoperability process.** This tool must provide specific wizards and automation facilities to simplify the definition of integration metamodels and metamodel mappings. This also reduces the potential errors that may be produced by a manual definition of the involved interoperability artifacts. We plan to automate the patterns that we are developing for the generation of an Integration Metamodel and the rules related to the identification of integration conflicts.
2. **An MDD interoperability suite for OO-Method.** This OO-Method suite will be implemented as an extension of the Eclipse UML2 tool. Thus, it will be possible to obtain an effective MDD solution that takes advantage of the UML extension capabilities and interchange standards and also take advantage of the current open-source MDD technologies, such as model transformation tools.

New MDD interoperability scenarios.

1. **Interoperability of multiple requirement modeling approaches.** We are currently working on a collaborative Project with the Universidad Politécnica de Cataluña. This project integrates different requirement modeling approaches to improve MDD processes. The objective of this project is the correct generation of Web services as well as the generation of testing mechanisms to assure that the generated services are aligned to the stakeholders' needs.
2. **Interoperability of MDD approaches.** This extends the proposed interoperability approach to support the coordination of multiple MDD approaches. We are currently working on the interoperability of a novel MDD approach related to early detection of defects in software systems with proposals related to model compilation. Thus, the quality of the software generated in MDD processes can be improved, and the cost of fixing software faults in late development stages can be reduced.

11.3. Publications

This section shows the publications that are related to the development of this thesis. These publications are divided in 17 international conferences, 1 national conference, 3 international journals, and 1 book chapter. The publications are the following:

1. Marín, B., **Giachetti, G.**, Pastor, O.: Intercambio de Modelos UML y OO-Method. X Workshop Iberoamericano de Ingeniería de Requisitos y Ambientes de Software (IDEAS). 2007.
2. **Giachetti G.**, Marín B., Condori-Fernández N., Molina J.C.: Updating OO-Method Function Points. QUATIC 2007: 55-64
3. Marín, B., **Giachetti, G.**, Pastor, O.: The Photography Agency: A case study of the OO-Method Approach. Technical Report DSIC-II/13/08, Universidad Politécnica de Valencia, Valencia, España (2008)
4. Marín B., **Giachetti G.**, Pastor O.: Una herramienta industrial para la medición del tamaño funcional de aplicaciones desarrolladas en entornos MDA. CIbSE 2008: 357-362.
5. Marín B., Pastor O., **Giachetti G.**: Automating the Measurement of Functional Size of Conceptual Models in an MDA Environment. PROFES 2008: 215-229
6. **Giachetti, G.**, Valverde, F., Pastor, O.: Improving Automatic UML2 Profile Generation for MDA Industrial Development. 4th International Workshop on Foundations and Practices of UML (FP-UML) – ER Workshop. Springer LNCS, 2008.
7. **Giachetti, G.**, Marín, B., Pastor, O.: Perfiles UML y Desarrollo Dirigido por Modelos: Desafíos y Soluciones para Utilizar UML como Lenguaje de Modelado Específico de Dominio. V Taller sobre Desarrollo de Software Dirigido por Modelos (DSDM) – Taller JISDB. 2008.
8. **Giachetti, G.**, Marín, B., Pastor, O.: Integración de UML y DSMLs en Entornos de Desarrollo Dirigido por Modelos. In: Proceedings of XII Conferencia Iberoamericana en Software Engineering CIbSE. 2009.
9. Marín B., **Giachetti G.**, Pastor O., Abran A.: Identificación de Defectos en Modelos Conceptuales utilizados en Entornos MDA. CIbSE. 2009.

10. Alencar, F., Pastor, O., Marín, B., **Giachetti, G.,** Castro, J.: Aligning Goal-Oriented Requirements Engineering and Model-Driven Development. In: Proceedings of 11th International Conference on Enterprise Information Systems (ICEIS). 2009.

11. **Giachetti, G.**, Marín, B., Pastor, O.: Using UML Profiles to Interchange DSML and UML Models. In: Proceedings of Third International Conference on Research Challenges in Information Science (RCIS). IEEE Computer Society, 2009.

12. **Giachetti, G.**, Marín, B., Pastor, O.: Using UML as a Domain-Specific Modeling Language: A Proposal for Automatic Generation of UML Profiles. In: Proceedings of CAiSE'09. Springer LNCS, 2009.

13. **Giachetti, G.**, Marín, B., Pastor, O.: Integration of Domain-Specific Modeling Languages and UML through UML Profile Extension Mechanism. In: International Journal of Computer Science & Applications (IJCSA). 2009.

14. Marín, B., **Giachetti, G.**, Pastor, O.: Applying a Functional Size Measurement Procedure for Defect Detection in MDD Environments 16th European Conference on Systems & Software Process Improvement and Innovation (EuroSPI). 2009

15. Alencar, F., Marín, B., **Giachetti, G.**, Pastor, O., Castro, J., Pimentel, J.H.: From *i** Requirements Models to Conceptual Models of a Model-Driven Development Process. In: Proceedings of 2nd Working Conference on The Practice of Enterprise Modeling (PoEM). Springer LNIBP, 2009.

16. **Giachetti, G.**, Alencar, F., Marín, B., Pastor, O., Castro, J.: Beyond Requirements: An Approach to Integrate i* and Model-Driven Development. In: Proceedings of XIII Conferencia Iberoamericana en Software Engineering (CIbSE 2010). 2010.

17. Pastor, O., **Giachetti, G.**: Linking Goal-Oriented Requirements and Model-Driven Development. Intentional Perpectives on Information Systems Engineering. Springer Book. 2010.

18. Marín B., **Giachetti G.**, Pastor O., Vos T.E.J., Abran A.: Evaluating the usefulness of a functional size measurement procedure to detect defects in MDD models. ESEM 2010.

19. Alencar, F., Marín, B., **Giachetti, G.**, Pastor, O., Castro, J., Franch, X., Pimentel, J.: From i* to OO-Method: Problems and Solutions Fourth

International i* Workshop (istar 2010) - CAiSE Workshops, vol. 586. CEUR Workshop Proceedings. 2010.

20. Giachetti, G., Albert, M., Marín, B., Pastor, O.: Linking UML and MDD Through UML Profiles: A Practical Approach based on the UML Association. Journal of Universal Computer Science (J.UCS). 2010

21. Marín, B., **Giachetti, G.**, Pastor, O., Abran, A.: A Quality Model for Conceptual Models of MDD Environments. Advances in Software Engineering 2010 Special Issue: New Generation of Software Metrics, ID 307391 (2010)

22. Marín, B., Giachetti, G., Pastor, O., Vos, T.E.J.: A Tool for Automatic Defect Detection in Models used in Model-Driven Engineering. 7th International Conference on the Quality of Information and Communications Technology (QUATIC), pp. 242–247. IEEE (2010)

23. Marín, B., Vos, T., Giachetti, G., Baars, A., Tonella, P.: Towards Testing Future Web Applications. 5th IEEE International Conference on Research Challenges in Information Science (RCIS). IEEE (2011)

Table 28. Publication Summary

Type	Publication Place	Number
National Conference	DSDM (JISBD Workshop)	1
International Conference	CAISE, RCIS, POEM, EuroSPI, ICEIS, ESEM, FP-UML (ER Workshop), QUATIC, PROFES, IDEAS, CIBSE, *i** Workshop (CAISE Workshop)	17
Journals	JUCS, IJCSA, ASE	3
Book Chapter	Springer Book	1

References

1. Selic, B.: The Pragmatics of Model-Driven Development. IEEE Software, vol. 20, pp. 19–25 (2003)

2. Völter, M., Stahl, T., Bettin, J., Haase, A., Helsen, S., Czarnecki, K.: Model-Driven Software Development: Technology, Engineering, Management. Wiley (2007)

3. Pohjonen, R., Kelly, S.: Domain-Specific Modeling. Dr. Dobb's Journal (2002)

4. Luoma, J., Kelly, S., Tolvanen, J.-P.: Defining Domain-Specific Modeling Languages: Collected Experiences. 4th OOPSLA Workshop on Domain-Specific Modeling (DSM'04), (2004)

5. OMG: UML 2.1.2 Superstructure Specification.

6. Yu, E.: Modelling Strategic Relationships for Process Reengineering, Ph.D. thesis, also Tech. Report DKBS-TR-94-6. Dept. of Computer Science. University of Toronto, Toronto, Canada (1995)

7. Ayala, C., Cares, C., Carvallo, J.P., Grau, G., Haya, M., Salazar, G., Franch, X., Mayol, E., Quer, C.: A Comparative Analysis of i*-Based Goal-Oriented Modelling Languages. International Workshop on Agent-Oriented Software Development Methodologies (AOSDM'05), at the SEKE Conference, pp. 657–663, Taipei, Taiwán; China. (2005)

8. Watson, A.: UML vs. DSLs: A false dichotomy. Object Management Group (2008)

9. Bruneliere, H., Cabot, J., Clasen, C., Jouault, F., Bezivin, J.: Towards Model Driven Tool Interoperability: Bridging Eclipse and Microsoft Modeling Tools. 6th European Conference on Modelling Foundations and Applications (ECMFA 2010), vol. LNCS 6138, pp. 32–47. Springer (2010)

10. OMG: MDA Guide Version 1.0.1. (2003)

11. Booch, G., Brown, A.W., Iyengar, S., Rumbaugh, J., Selic, B.: An MDA Manifesto. Business Process Trends/MDA Journal (2004)

References

12. Pastor, O., Molina, J.C.: Model-Driven Architecture in Practice: A Software Production Environment Based on Conceptual Modeling. Springer, New York (2007)

13. Kent, S.: Model Driven Engineering. Integrated Formal Methods (IFM), pp. 286–298. Springer, London, UK (2002)

14. Albert, M.: Tratamiento de Asociaciones en Entornos de Producción Automática de Código. Departamento de Sistemas Informáticos y Computación, vol. Ph.D Thesis. . Universidad Politécnica de Valencia (2006)

15. Rolland, C., Prakash, N., Benjamen, A.: A multi-model view of process modeling. Requirements Engineering 4, 169–187 (1999)

16. Pastor, O., Gómez, J., Insfrán, E., Pelechano, V.: The OO-Method Approach for Information Systems Modelling: From Object-Oriented Conceptual Modeling to Automated Programming. Information Systems, vol. 26, pp. 507–534. Elsevier Science (2001)

17. Selic, B.: A Systematic Approach to Domain-Specific Language Design Using UML. 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC), pp. 2–9 (2007)

18. Giachetti, G., Marin, B., Pastor, O.: Integration of Domain-Specific Modeling Languages and UML through UML Profile Extension Mechanism International Journal of Computer Science and Applications 6, 145–174 (2009)

19. Wimmer, M., Schauerhuber, A., Strommer, M., Schwinger, W., Kappel, G.: A Semi-automatic Approach for Bridging DSLs with UML. 7th OOPSLA Workshop on Domain-Specific Modeling (DSM), pp. 97–104 (2007)

20. Alencar, F., Marín, B., Giachetti, G., Pastor, O., Castro, J., Pimentel, J.H.: From i* Requirements Models to Conceptual Models of a Model Driven Development Process. 2nd Working Conference on The Practice of Enterprise Modeling (PoEM 2009), vol. LNIBP 39, pp. 99–114. Springer (2009)

21. Giachetti, G., Alencar, F., Marín, B., Pastor, O., Castro, J.: Beyond Requirements: An Approach to Integrate i* and Model-Driven Development. XIII Conferencia Iberoamericana en Software Engineering (CIbSE 2010), (2010)

22. Falleri J.-R., H.M., Mathieu L., Nebut C. : Metamodel Matching for Automatic Model Transformation Generation. 11th International Conference on Model

Driven Engineering Languages and Systems (MoDELS 2008), vol. LNCS 5301, pp. 326–340. Springer (2008)

23. http://pros.upv.es/

24. Universidad Politécnica de Valencia, http://www.upv.es/

25. Generalitat Valenciana, http://www.gva.es/

26. ProS Research Center, http://www.pros.upv.es/index.php/es/proyectos/149-prosreq

27. Ministerio de Ciencia e Innovación, http://www.micinn.es/

28. GESSI, http://www.essi.upc.edu/~gessi/

29. IEEE: Standard Glossary of Software Engineering Terminology. Standard 729-1983. IEEE Computer Society (1983)

30. Staron, M., Wohlin, C.: An Industrial Case Study on the Choice Between Language Customization Mechanisms. Product-Focused Software Process Improvement (PROFES), pp. 177–191. Springer (2006)

31. France, R.B., Ghosh, S., Dinh-Trong, T., Solberg, A.: Model-driven development using uml 2.0: Promises and pitfalls. IEEE Computer, vol. 39, pp. 59–66 (2006)

32. Pastor, O., Giachetti, G.: Linking Goal-Oriented Requirements and Model-Driven Development. In: Nurcan, S., Salinesi, C., Souveyet, C., Ralyté, J. (eds.) Intentional Perpectives on Information Systems Engineering, pp. 257–276. Springer-Verlag (2010)

33. Bruck, J., Hussey, K.: Customizing UML: Which Technique is Right for You? IBM (2007)

34. http://www.omg.org/

35. OMG: UML 2.3 Infrastructure Specification. (2010)

36. OMG: UML 2.1.1 Infrastructure Specification.

37. OMG: Object Constraint Language 2.2 Specification. (2010)

38. OMG: MOF 2.0 Core Specification. (2006)

References

39. OMG: UML 2.2 Infrastructure Specification.

40. OMG: XMI 2.1.1 Specification.

41. OMG: UML 2.2 Superstructure Specification.

42. Pardillo, J.: A Systematic Review on the Definition of UML Profiles. 13th International Conference on Model Driven Engineering Languages and Systems (MODELS 2010), vol. LNCS 6394, pp. 407–422. Springer-Verlag (2010)

43. OMG: Catalog of UML Profile Specifications.

44. Moreno, N., Fraternali, P., Vallecillo, A.: WebML Modeling in UML. IET Software, vol. 1, pp. 67–80 (2007)

45. Molina, J.C., Pastor, O.: MDA, OO-Method y la Tecnología OLIVANOVA Model Execution. I Taller sobre Desarrollo de Software Dirigido por Modelos, MDA y Aplicaciones (DSDM'04), (2004)

46. OMG: SysML Final Adopted Specification v1.0.

47. http://www.omgsysml.org/

48. Harel, D., Rumpe, B.: Meaningful Modeling: What's the Semantics of "Semantics"? IEEE Computer, vol. 37, pp. 64–72 (2004)

49. Fabro, M.D.D., Valduriez, P.: Towards the efficient development of model transformations using model weaving and matching transformations. Software and Systems Modeling 8, 305–324 (2009)

50. http://www.eclipse.org/gmf/

51. Fuentes-Fernández, L., Vallecillo, A.: An Introduction to UML Profiles. The European Journal for the Informatics Professional (UPGRADE), vol. 5, pp. 5–13 (2004)

52. http://www.omg.org/mof/

53. Selic, B.: The Theory and Practice of Modeling Language Design for Model-Based Software Engineering—A Personal Perspective. 4th Summer School on Generative and Transformational Techniques in Software Engineering (GTTSE 2009), vol. LNCS 6491, pp. 290–321. Springer-Verlag, Braga, Portugal (2009)

54. Lagarde, F., Espinoza, H., Terrier, F., Gérard, S.: Improving UML Profile Design Practices by Leveraging Conceptual Domain Models. 22th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 445–448 (2007)

55. Mallet, F., Lagarde, F., André, C., Gérard, S., Terrier, F.: An automated process for implementing multilevel domain models. 2nd International Conference on Software Language Engineering (SLE'09), vol. LNCS 5969, pp. 314–333. Springer-Verlag (2009)

56. Robert, S., Gérard, S., Terrier, F., Lagarde, F.: A Lightweight Approach for Domain-Specific Modeling Languages Design. 35th Euromicro Conference on Software Engineering and Advanced Applications, pp. 155–161. IEEE (2009)

57. http://www.eclipse.org/emf/

58. http://www.eclipse.org/m2m/atl/

59. OMG: QVT 1.0 Specification. (2008)

60. Henderson-Sellers, B., Gonzalez-Perez, C.: Uses and Abuses of UML Profile Extension Mechanism in UML 1.x and 2.0. In: al, O.N.e. (ed.) MoDELS 2006, vol. LNCS 4199, pp. 16–26. Springer-Verlag Berlin Heidelberg (2006)

61. Pastor, O., Gómez, J., Insfrán, E., Pelechano, V.: The OO-Method Approach for Information Systems Modelling: From Object-Oriented Conceptual Modeling to Automated Programming. Information Systems 26, 507–534 (2001)

62. Pastor, O., Molina, J.C., Iborra, E.: Automated production of fully functional applications with OlivaNova Model Execution. ERCIM News (2004)

63. Gómez, J., Insfrán, E., Pelechano, V., Pastor, O.: The Execution Model: a component-based architecture to generate software components from conceptual models. Workshop on Component-based Information Systems Engineering, Pisa, Italia (1998)

64. http://www.care-t.com/

65. OMG: UML 2.1.2 Superstructure Specification. (2007)

66. Molina, P.: Especificación de interfaz de usuario: De los requisitos a la generación automática. Universidad Politécnica de Valencia, Valencia, España (2003)

67. Martínez, A.: Conceptual Schemas Generation from Organizational Models in an Automatic Software Production Process. Ph.D. thesis. Universidad Politécnica de Valencia, Valencia, Spain (2008)

68. Yu, E.: Modelling Strategic Relationships for Process Reengineering. PhD Thesis. University of Toronto, Toronto, Canada (1995)

69. Yu, E.: Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. 3rd IEEE Int. Symp. on Requirements Engineering (RE'97), pp. 226–235 Washington D.C., USA (1997)

70. Yu, E., Liu, L., Li, Y.: Modelling Strategic Actor Relationships to Support Intellectual Property Management. 20th International Conference on Conceptual Modeling (ER-2001), vol. LNCS 2224, pp. 164–178 Spring Verlag, Yokohama, Japan (2001)

71. Elvesæter, B., Hahn, A., Berre, A.-J., Neple, T.: Towards an Interoperability Framework for Model-Driven Development of Software Systems. 1st Interntional Conference on Interoperability of Enterprise Software and Applications (INTEROP-ESA'05), pp. 409–420. Springer (2006)

72. Kitchenham, B.A.: Guidelines for performing systematic literature reviews in software engineering. EBSE Technical Report EBSE-2007-001 (2007)

73. Petticrew, M., Roberts, H.: Systematic Reviews in the Social Sciences: A Practical Guide. (2005)

74. Roser, S., Bauer, B.: Improving Interoperability in Collaborative Modelling. 3rd International Conference on Interoperability of Enterprise Software and Applications (I-ESA 2007), vol. Enterprise Interoperability II, pp. 139–150. Springer-Verlag (2007)

75. Fabro, M.D.D., Valduriez, P.: Semi-automatic model integration using matching transformations and weaving models. 2007 ACM symposium on Applied computing (SAC '07), pp. 963–970. ACM New York (2007)

76. Tran, H., Zdun, U., Dustdar, S.: View-Based Reverse Engineering Approach for Enhancing Model Interoperability and Reusability in Process-Driven SOAs 10th International Conference on Software Reuse (ICSR 2008), vol. LNCS 5030, pp. 233–244. Springer (2008)

77. Demirezen, Z., Sun, Y., Gray, J., Jouault, F.: Enabling tool reuse and interoperability through model-driven engineering. Journal of Computational Methods in Science and Engineering 20, 187–202 (2010)

78. Kappel, G., Wimmer, M., Retschitzegger, W., Schwinger, W.: Leveraging Model-Based Tool Integration by Conceptual Modeling Techniques. The Evolution of Conceptual Modeling, vol. LNCS 6520, pp. 254–284. Springer-Verlag (2011)

79. Fabro, M.D.D., Valduriez, P.: Towards the Efficient Development of Model Transformations using Model Weaving and Matching Transformations. Software and Systems Modeling (SoSyM) 8, 305–324 (2009)

80. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: ATL: A model transformation tool. Science of Computer Programming 72, 31–39 (2008)

81. Klar, F., Rose, S., Schürr, A.: A Meta-model-Driven Tool Integration Development Process. 2nd International United Information Systems Conference (UNISCON'2008), vol. LNBIP 5, pp. 201–212. Springer (2008)

82. Guerra, E., Lara, J.d., Orejas, F.: Inter-modelling with patterns. Software and Systems Modeling (SoSym) (2011)

83. Seifert, M., Wende, C., Aßmann, U.: Anticipating Unanticipated Tool Interoperability using Role Models. 1st Workshop on Model Driven Interoperability, pp. 52–60. ACM (2010)

84. Insfrán, E., Pelechano, V., Pastor, O.: Conceptual Modeling in the eXtreme Information & Software Technology - INFSOF 44, 659–669 (2002)

85. Agostinho, C., Correia, F., Jardim-Goncalves, R.: Interoperability of Complex Business Networks by Language Independent Information Models. 17th ISPE International Conference on Concurrent Engineering (CE 2010), vol. Advanced Concurrent Engineering, pp. 111–124. Springer-Verlag (2011)

86. Radjenovic, A., Paige, R.F.: Behavioural Interoperability to Support Model-Driven Systems Integration. 1st Workshop on Model Driven Interoperability (MDI 2010), pp. 98–107. ACM (2010)

87. Polgár, B., Ráth, I., Szatmári, Z., Horváth, Á., Majzik, I.: Model-based Integration, Execution and Certification of Development Tool-chains. Second European Workshop on Model Driven Tool and Process Integration (MDTPI), vol. WP09-05, pp. 35-46. CTIT Workshop Proceedings Series (2009)

88. Kolovos, D., Paige, R., Rose, L., Polack, F.: The Epsilon Book. Eclipse Foundation (2010)

89. Crnkovic, I., Malavolta, I., Muccini, H.: A Model-Driven Engineering Framework for Component Models Interoperability. International Symposium on Component Based Software Engineering (CBSE), vol. LNCS 5582, pp. 36–53. Springer-Verlag (2009)

90. Ziemann, J., Ohren, O., Jäkel, F.-W., Kahl, T., Knothe, T.: Achieving Enterprise Model Interoperability Applying a Common Enterprise Metamodel. 2nd International Conference on Interoperability of Enterprise Software and Applications (I-ESA 2006), vol. Enterprise Interoperability, pp. 199–208. Springer-Verlag, Bordeaux, France (2007)

91. Jankovic, M., Ivezic, N., Knothe, T., Marjanovic, Z., Snack, P.: A Case Study in Enterprise Modelling for Interoperable Cross-Enterprise Data Exchange. 3rd International Conference on Interoperability of Enterprise Software and Applications (I-ESA 2007), vol. Enterprise Interoperability II, pp. 541–552. Springer-Verlag (2007)

92. Ohren, O.P., Chen, D., Grangel, R., Jaekel, F.-W., Karlsen, D., Knothe, T., Rolfsen, R.K.: ATHENA-A1, Deliverable DA1.5.2: Report on Methodology description and guidelines definition. (2005)

93. Baumgart, A.: A common meta-model for the interoperation of tools with heterogeneous data models. 3rd European Workshop on Model Driven Tool and Process Integration (MDTPI), (2010)

94. Mahé, V., Brunelière, H., Jouault, F., Bézivin, J., Talpin, J.-P.: Model-Driven Interoperability of Dependencies Visualizations. 3rd European Workshop on Model Driven Tool and Process Integration (MDTPI), (2010)

95. Berger, S., Grossmann, G., Stumptner, M., Schrefl, M.: Metamodel-Based Information Integration at Industrial Scale. 13th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2010), vol. LNCS 6395, pp. 153–167. Springer-Verlag (2010)

96. Vallecillo, A.: On the Combination of Domain Specific Modeling Languages. 6th European Conference on Modelling Foundations and Applications (ECMFA 2010), vol. LNCS 6138, pp. 305–320. Springer (2010)

97. Coutinho, L., Brandão, A., Sichman, J., Boissier, O.: Model-Driven Integration of Organizational Models. (AOSE 2008), vol. LNCS 5386, pp. 1–15. Springer (2009)

98. Moreno, N., Vallecillo, A.: Towards Interoperable Web Engineering Methods. Journal of the American Society for Information Science and Technology 59, 1073–1092 (2008)

99. Giachetti, G., Alencar, F., Franch, X., Marín, B., Pastor, O.: Technical Report ProS-TR-2011-07: Automatic Verification of Requirement Models for Their Interoperability in Model-Driven Development Processes. Universidad Politécnica de Valencia (2011)

100. Biehl, M., Sjöstedt, C.-J., Törngren, M.: A Modular Tool Integration Approach - Experiences from two Case Studies. 3rd European Workshop on Model Driven Tool and Process Integration (MDTPI), (2010)

101. Jouault, F., Guéguen, T.: Integration by Model-driven Virtual Tool. 2nd European Workshop on Model Driven Tool and Process Integration (MDTPI 2009), vol. WP09-05. CTIT Workshop Proceedings Series (2009)

102. Brambilla, M., Fraternali, P., Tisi, a.M.: A Transformation Framework to Bridge Domain Specific Languages to MDA. Models in Software Engineering Workshops and Symposia at MODELS 2008, vol. LNCS 5421, pp. 167–180. Springer-Verlag (2008)

103. Lukácsy, G., Szeredi, P., Benkő, T.: Towards automatic semantic integration. 3rd International Conference on Interoperability of Enterprise Software and Applications (I-ESA 2007), vol. Enterprise Interoperability II, pp. 795–806. Springer-Verlag (2007)

104. Hein, C., Ritter, T., Wagner, M.: Model-Driven Tool Integration with ModelBus. 1st International Workshop on Future Trends of Model-Driven Development (FTMDD 2009), (2009)

105. Blanc, X., Gervais, M.-P., Sriplakich, P.: Model Bus: Towards the Interoperability of Modelling Tools. Model-Driven Architecture: Foundations and Applications (MDAFA), vol. LNCS 3599, pp. 17–32. Springer-Verlag Berlin Heidelberg (2004)

106. Höfferer, P.: Achieving Business Process Model Interoperability Using Metamodels and Ontologies. 15th European Conference on Information Systems (ECIS 2007), (2007)

107. Sunindyo, W.D., Moser, T., Winkler, D., Biffl, S.: A Process Model Discovery Approach for Enabling Model Interoperability in Signal Engineering. 1st Workshop on Model Driven Interoperability, pp. 15–21. ACM (2010)

108. Berre, A.-J., Liu, F., Xu, J., Elvesaeter, B.: Model Driven Service Interoperability through use of Semantic Annotations. International Conference on Interoperability for Enterprise Software and Applications China (IESA '09), pp. 90–96 IEEE (2009)

109. Barnickel, N., Fluegge, M.: Towards a Conceptual Framework for Semantic Interoperability in Service Oriented Architectures. 1st International Conference on Intelligent Semantic Web-Services and Applications. ACM (2010)

110. Opdahl: Incorporating UML Class and Activity Constructs into UEML. International conference on Advances in conceptual modeling: applications and challenges - ER 2010 Workshops, vol. LNCS 6413, pp. 244–254. Springer-Verlag (2010)

111. Anaya, V., Berio, G., Harzallah, M., Heymans, P., Matulevicius, R., Opdahl, A.L., Panetto, H., Verdecho, M.J.: The Unified Enterprise Modelling Language - Overview and further work. Computers in Industry 61, 99–111 (2010)

112. Costa, R., Garcia, O., Nuñez, M., Maló, P., Gonçalves, R.: Integrated solution to support enterprise interoperability at the business process level on e-Procurement. 3rd International Conference on Interoperability of Enterprise Software and Applications (I-ESA 2007), vol. Enterprise Interoperability II, pp. 89–100. Springer (2007)

113. Kappel, G., Kapsammer, E., Kargl, H., Kramler, G., Reiter, T., Retschitzegger, W., Schwinger, W., Wimmer, M.: Lifting Metamodels to Ontologies: A Step to the Semantic Integration of Modeling Languages. 9th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2006), vol. LNCS 4199, pp. 528–542. Springer (2006)

114. IEEE: IEEE Standard Computer Dictionary: Compilation of IEEE Standard Computer Glossaries. (1990)

115. Nilsson, M., Baker, T., Johnston, P.: Interoperability Levels for Dublin Core Metadata. Dublin Core Metadata Initiative (2009)

116. Wang, W., Tolk, A., Wang, W.: The levels of conceptual interoperability model: applying systems engineering principles to M&S. 2009 Spring Simulation

Multiconference (SpringSim '09). Society for Computer Simulation International (2009)

117. E.U.: European Interoperability Framework for European Public Services (EIF) Version 2.0 (Draft). European Commission (2008)

118. A.W.P.: Levels of Information Systems Interoperability (LISI). Department of Defense - United Sates of America (1998)

119. Haslhofer, B., Klas, W.: A survey of techniques for achieving metadata interoperability. ACM Computing Surveys (CSUR) 42, (2010)

120. Ouksel, A.M., Sheth, A.: Semantic interoperability in global information systems. ACM SIGMOD 28, 5–12 (1999)

121. Sarantis, D., Charalabidis, Y., Psarras, J.: Towards Standardising Interoperability Levels for Information Systems of Public Administrations. eJETA Special Issue on "Interoperability for Enterprises and Administrations Worldwide". Yannis Charalabidis, Hervé Panetto, Euripidis Loukis, Kai Mertins (2008)

122. OMG: UML 2.3 Superstructure Specification. (2010)

123. Abdulhadi, S.: *i** Guide version 3.0. (2007)

124. http://www.w3.org/XML/

125. http://www.eclipse.org/modeling/

126. http://www.eclipse.org/uml2/

127. Giachetti, G., Valverde, F., Pastor, O.: Improving Automatic UML2 Profile Generation for MDA Industrial Development. In: Workshops, E. (ed.) 4th International Workshop on Foundations and Practices of UML (FP-UML) – ER Workshop, vol. LNCS 5232, pp. 113–122. Springer (2008)

128. White, S.A., Miers, D.: BPMN Modeling and Reference Guide: Understanding and Using BPMN. Future Strategies Inc., FL, USA (2008)

129. Henderson-Sellers, B.: On the Challenges of Correctly Using Metamodels in Software Engineering. 6th Conference on Software Methodologies, Tools, and Techniques (SoMeT), pp. 3–35 (2007)

130. Abouzahra, A., Bézivin, J., Fabro, M.D.D., Jouault, F.: A Practical Approach to Bridging Domain Specific Languages with UML profiles. Best Practices for Model Driven Software Development (OOPSLA'05), (2005)

131. OMG: UML 2.1.2 Infrastructure Specification.

132. Marín, B., Giachetti, G., Pastor, O.: Intercambio de Modelos UML y OO-Method. X Workshop Iberoamericano de Ingeniería de Requisitos y Ambientes de Software (IDEAS), pp. 283–296, Isla Margarita, Venezuela (2007)

133. Queralt, A., Teniente, E.: Decidable Reasoning in UML Schemas with Constraints. 20th Conference on Advanced Information Systems Engineering (CAiSE'08), vol. LNCS 5074, pp. 281–295. Springer (2008)

134. http://www.eclipse.org/modeling/mdt/

135. Giachetti, G., Marín, B., Pastor, O.: Using UML as a Domain-Specific Modeling Language: A Proposal for Automatic Generation of UML Profiles. 21st International Conference on Advanced Information Systems (CAiSE 2009), vol. LNCS 5565, pp. 110–124. Springer (2009)

136. Giachetti, G., Marín, B., Pastor, O.: Using UML Profiles to Interchange DSML and UML Models. Third International Conference on Research Challenges in Information Science (RCIS), pp. 385–394. IEEE Computer Society (2009)

137. Giachetti, G., Marín, B., Condori-Fernández, N., Molina, J.C.: Updating OO-Method Function Points. 6th IEEE International Conference on the Quality of Information and Communications Technology (QUATIC 2007), pp. 55–64, Lisboa, Portugal (2007)

138. Marín, B., Giachetti, G., Pastor, O.: Automating the Measurement of Functional Size of Conceptual Models in an MDA Environment. Product-Focused Software Process Improvement (PROFES), vol. LNCS, pp. 215–229. Springer (2008)

139. Marín, B., Giachetti, G., Pastor, O.: Una Herramienta Industrial para la Medición del Tamaño Funcional de Aplicaciones Desarrolladas en Entornos MDA. XI Workshop Iberoamericano de Ingeniería de Requisitos y Ambientes de Software (IDEAS), Recife, Brasil (2008)

140. ISO/IEC, (20926): Software Engineering – IFPUG 4.1 Unadjusted Functional Size Measurement Method – Counting Practices Manual. (2003)

141. http://www.omg.org/mda/committed-products.htm

142. Milicév, D.: On the Semantics of Associations and Association Ends in UML. IIEE Transactions on Software Engineering 33, (2007)

143. Graham, I., Bischof, J., Henderson-Sellers, B.: Associations considered a bad thing. Journal of Object-oriented Programming 9, 1–48 (1997)

144. Snoeck, M., Dedene, G.: Core modeling concepts to define aggregation. L'objet 7, 281–306 (2001)

145. Albert, M., Pelechano, V., Fons, J., Ruiz, M., Pastor, O.: Implementing UML Association, Aggregation, and Composition. A Particular Interpretation Based on a Multidimensional Framework 15th Conference on Advanced Information Systems Engineering (CAISE'03), pp. 143–158 (2003)

146. Guéhéneuc, Y., Albin-Amiot, H.: Recovering binary class relationships: Putting icing on the UML cake Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA'04), pp. 301–314 (2004)

147. Diskin, Z., Dingel, J.: Mappings, Maps and Tables: Towards Formal Semantics for Associations in UML2. MoDELS 2006, pp. 230–244. LNCS (2006)

148. Génova, G., Llorens, J., Fuentes, J.M.: UML Associations: A Structural and Contextual View Journal of Object Technology 3, (2004)

149. OMG: UML 1.4.2 Specification.

150. Henderson-Sellers, B., & Barbier, F.: What is this thing called aggregation? In: In A. C. Wills, J.B., R. Mitchell, & B. Meyer (Eds.) (ed.) TOOLS 29, pp. 216–230. IEEE Computer Society (1999)

151. Henderson-Sellers, B., Barbier, F.: Black and white diamonds. In: (Eds.), I.R.F.B.R. (ed.) UML'99, pp. 550–565. LNCS (1999)

152. Opdahl, A.L., Henderson-Sellers, B., Barbier, F.: Ontological analysis of whole-part relationships in OO-models. Information and Software Technology 387–399 (2001)

153. Barbier, F., Henderson-Sellers, B., Le Parc-Lacayrelle, A., Bruel, J.-M.: Formalization of the Whole-Part Relationship in the Unified Modeling Language. IEEE Transactions on Software Engineering 29, 459–470 (2003)

154. Belloir, N., Bruel, J.-M., Barbier, F.: Whole-Part Relationships for Software Component Combination. 29th EUROMICRO Conference (EUROMICRO'03), pp. 86–91. IEEE Computer Society (2003)

155. Stevens, P.: On the interpretation of binary associations in the Unified Modelling Language. Journal on Software and System Modeling 1, 68–79 (2002)

156. Akehurst, D.H., Howells, W.G.J., McDonald-Maier, K.D: Implementing Associations: UML 2.0 to Java 5. Journal of Software and Systems Modeling 6, 1–33 (2006)

157. Gessenharter, D.: Mapping the UML2 Semantics of Associations to Java Code Generation Model. 11th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2008), vol. LNCS 5301, pp. 813–827 (2008)

158. Marín, B., Giachetti, G., Pastor, O.: The Photography Agency: A case study of the OO-Method Approach. Technical Report DSIC-II/13/08. Universidad Politécnica de Valencia (2008)

159. Sunyé, G., Pennaneac'h, F., Ho, W.-M., Guennec, A.L., Jézéquel, J.-M.: Using UML Action Semantics for Executable Modeling and Beyond. CAiSE 2001, vol. LNCS 2068, pp. 433–447. Springer, Interlaken, Switzerland (2001)

160. Budinsky, F., Brodsky, S.A., Merks, E.: Eclipse Modeling Framework. Pearson Education (2003)

161. Lamsweerde, A.v.: Systematic Requirements Engineering - From System Goals to UML Models to Software Specifications. Wiley (2008)

162. Nuseibeh, B., Easterbrook, S.M.: Requirements Engineering: A Roadmap. In The Future of Software Engineering IEEE Computer Society Press (2000)

163. Shuichiro, Y., Haruhiko, K., Karl, C., Steven, B.: Goal Oriented Requirements Engineering: Trends and Issues. IEICE - Trans. Inf. Syst. E89-D, 2701–2711 (2006)

164. Lamsweerde, A.v.: Goal-oriented requirements engineering: a roundtrip from research to practice. 12th IEEE Joint International Requirements Engineering Conference, pp. 4–8. IEEE Computer Science Press (2004)

165. Lamsweerde, A.v.: Goal-oriented requirements engineering: A guided tour. 5th IEEE International Symposium on Requirements Engineering (RE'01), (2001)

166. Rolland, C., Prakash, N.: From conceptual modelling to requirements engineering. Annals of Soft. Eng 10, 151–176 (2000)

167. Rolland, C., Souveyet, C., Achour, C.B.: Guiding Goal Modelling Using Scenarios. IEEE Transactions on Software Engineering (IEEE TSE), Special Issue on Scenario Management 24, 1055–1071 (1998)

168. Maiden, N.A.M., Jones, S.V., S.Manning, Greenwood, J., Renou, L.: Model-Driven Requirements Engineering: Synchronising Models in an Air Traffic Management Case Study. CaiSE'2004, pp. 368–383. Springer-Verlag LNCS 3084 (2004)

169. Jouault, F., Kurtev, I.: Transforming Models with ATL. Satellite Events at the MoDELS 2005 Conference, vol. LNCS 3844, pp. 128–138 Springer (2006)

170. Martínez, A., Castro, J., Pastor, O., Estrada, H.: Closing the gap between Organizational Modeling and Information System Modeling. 6th Workshop on Requirements Engineering (WER'03), pp. 93–108, Piracicaba, Brazil (2003)

171. http://istar.rwth-aachen.de/, last Accessed October 2009

172. Lucena, M., Santos, E., Silva, M.J., Silva, C., Alencar, F., Castro, J.F.B.: Towards a Unified Metamodel for i*. 2nd IEEE Int. Conference on Research Challenges in Information Science (RCIS 2008), pp. 237–246 IEEE (2008)

173. Spanoudakis, G., Zisman, A: Software Traceability: A Roadmap. Handbook of Software Engineering and Knowledge Engineering, Vol. III: Recent Advancements, pp. 395–428. World Scientific Publishing Co. (2005)

174. Gotel, O., Finkelstein, A.: An Analysis of the Requirements Traceability Problem. 1st International Conference on Requirements Engineering (ICRE'94), pp. 94–101, Colorado Springs (1994)

175. Santander, V., Castro, J.: Deriving Use Cases from Organizational Modeling. In: 10th Anniversary IEEE Joint International Conference on Requirements Engineering (RE 2002), pp. 32–42. (Year)

176. Liu, L., Yu, E.: Designing Information Systems in Social Context: A Goal and Scenario Modeling Approach. Information Systems 29, 187–203 (2004)

177. Alencar, F.M.R., Pedroza, F.P., Castro, J., Amorim, R.C.O.: New Mechanisms for the Integration of Organizational Requirements and Object Oriented Modeling. 6th Workshop on Requirements Engineering (WER'03), pp. 109–123, Brasil. Piracicaba-SP (2003)

178. Cabot, J., Yu, E.: Improving Requirements Specifications in Model-Driven Development Processes. 1st Int. Workshop on Challenges in Model-Driven Software Engineering (MoDELS'08), (2008)

179. Dardenne, A., Van Lamsweerde, A., Fickas, S.: Goal-Directed Requirements Acquisition. Science of Computer Programming 20, 3–50 (1993)

180. Eric Yu, P.G., Neil Maiden and John Mylopoulos: Social Modeling for Requirements Engineering (2011)

181. ISO: International vocabulary of basic and general terms in metrology (VIM). International Organization for Standarization (2004)

182. Boehm, B.W.: Software Engineering Economics. Prentice-Hall Inc., Englewood Cliff, New Jersey (1981)

183. Habra, N., Abran, A., Lopez, M., Sellami, A.: A framework for the design and verification of software measurement methods. Journal of Systems and Software 81, 633–648 (2008)

184. Loniewski, G., Insfran, E., Abrahao, S.: A Systematic Review of the Use of Requirement Engineering Techniques in Model-Driven Development. 13th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2010), vol. LNCS 6395, pp. 213–227. Springer-Verlag (2010)

185. Lu, C.-W., Chang, C.-H., Chu, W.C., Cheng, Y.-W., Chang, H.-C.: A Requirement Tool to Support Model-Based Requirement Engineering. 32nd Computer Software and Applications Conference (COMPSAC '08), pp. 712–717 IEEE (2008)

186. Mellor, S.J., Scott, K., Uhl, A., Weise, D.: MDA Distilled: Principles of Model-Driven Architecture. Addison-Wesley Professional (2004)

187. Gross, D., Yu, E.: From Non-Functional Requirements to Design through Patterns. Requirements Engineering Journal 6, 18–36 (2001)

188. Hailpern, B., Tarr, P.: Model-driven development: The good, the bad, and the ugly. IBM Systems Journal 45, (2006)

189. Laguna, M.A., Gonzalez-Baixauli, B.: Requirements variability models: metamodel based transformations. Symposia on Metainformatics (MIS '05). ACM (2005)

190. Lapouchnian, A., Yu, Y., Liaskos, S., Mylopoulos, J.: Requirements-driven design of autonomic application software. Conference of the Center for Advanced Studies on Collaborative Research (CASCON 2006). ACM (2006)

191. Lamsweerde, E.L.a.A.v.: Deriving Operational Software Specifications from System Goals. 10th ACM SIGSOFT Symp. on the Foundations of Software Engineering (FSE'10). ACM (2002)

192. Pardillo, J., Molina, F., Cachero, C., Toval, A.: A UML Profile for Modelling Measurable Requirements. In: 4th International Workshop on Foundations and Practices of UML (FP-UML) – ER Workshop, pp. 123–132. Springer-Verlag, (Year)

193. Giorgini, P., Rizzi, S., Garzetti, M.: Goal-oriented Requirement Analysis for Data Warehouse Design. 8th Int. Workshop on Data Warehousing and OLAP, pp. 47–56. ACM Press (2005)

194. Amyot, D., Ghanavati, S., Horkoff, J., Mussbacher, G., Peyton, L., Yu, E.: Evaluating goal models within the goal-oriented requirement language. To appear In: International Journal of Intelligent Systems (IJIS) (2010)

195. Tong, Y., Fangjun, W., Chengzhi, G.: A comparison of metrics for UML class diagrams. ACM SIGSOFT Software Engineering Notes 29, (2004)

196. Genero, M., Piattini, M., Calero, C.: A Survey of Metrics for UML Class Diagrams. Journal of Object Technology 4, (2005)

197. Marín, B., Giachetti, G., Pastor, O.: Applying a Functional Size Measurement Procedure for Defect Detection in MDD Environments 16th European Conference on Systems & Software Process Improvement and Innovation (EuroSPI 2009). Springer (2009)

198. Basili, V., Caldeira, G., Rombach, H.D.: The Goal Question Metric Approach. Encyclopedia of Software Engineering, Wiley (1994)

199. Basili, V., Rombach, H.: The TAME Project: Towards Improvement Oriented Software Environments. IEEE Transactions on Software Engineering 14, 758–773 (1988)

200. Franch, X.: A Method for the Definition of Metrics over i* Models. 21st International Conference on Advanced Information Systems (CAiSE 2009), pp. 201-215. Springer-Verlag LNCS (2009)

References

201. Franch, X., Grau, G.: Towards a Catalogue of Patterns for Defining Metrics over i* Models. 20th International Conference on Advanced Information Systems (CAiSE 2008), pp. 197–212. Springer (2008)

202. Amyot, D., Horkoff, J., Gross, D., Mussbacher, G.: A Lightweight GRL Profile for i* Modeling. 3rd International Workshop on Requirements, Intentions and Goals in Conceptual Modeling (RIGIM) - ER Workshops, vol. LNCS 5833, pp. 254–264. Springer-Verlag (2009)

203. ISO/IEC: ISO/IEC 9126-1, Software Eng. – Product Quality – Part 1: Quality model. (2001)

204. Wohlin, C., Runeson, P., Host, M., Ohlsson, M., Regnell, B., Wesslén, A.: Experimentation in Software Engineering - An Introduction. Kluwer Academic (2000)

205. OMG: Business Process Modeling Notation version 1.1. (2008)

206. Giachetti, G., Albert, M., Marín, B., Pastor, O.: Linking UML and MDD Trough UML Profiles: A Practical Approach based on the UML Association. Journal of Universal Computer Science (J.UCS) 16, (2010)

207. Chung, L., Nixon, B., Yu, E., Mylopoulos, J.: Non-Functional Requirements in Software Engineering. Springer (1999)

208. Franch, X.: Incorporating Modules into the i* Framework. 22nd International Conference on Advanced Information Systems (CAiSE 2010), vol. LNCS 6051, pp. 439–454. Springer-Verlag Berlín Heidelberg, Hammamet, Tunisia (2010)

209. Amyot, D.: New draft Recommendation Z.151: User Requirements Notation (URN). (2008)

210. http://ais.affiniscape.com/displaycommon.cfm?an=1&subarticlenbr=279 last updated August 20, 2009

211. Takeda, H., Veerkamp, P., Tomiyama, T., Yoshikawam, H.: Modeling Design Processes AI Magazine Winter 37–48 (1990)

212. Giachetti, G., Marín, B., Pastor, O.: Perfiles UML y Desarrollo Dirigido por Modelos: Desafíos y Soluciones para Utilizar UML como Lenguaje de Modelado Específico de Dominio. V Taller sobre Desarrollo de Software Dirigido por Modelos (DSDM), (2008)

213. Dobing, B., Parsons, J.: How Used is UML. Communications of the ACM 49, 109–113 (2006)

Appendix I.
Transformation of *i** Models into MDD-Oriented Models

A good understanding of system requirements has a high impact in the successful development of software products. Therefore, an appropriate requirement model must provide a comprehensive structure for what must be elicited, evaluated, specified, consolidated, and modified, instead of just providing facilities for software specifications. Since there is a well-known gap between requirements specifications and final software products, we propose the integration of Goal-Oriented Requirements Engineering (GORE) and Model-Driven Development (MDD) to solve this gap. This appendix shows the elements that are the foundations to achieve the integration of i and OO-Method presented in this thesis as proof of concepts for the interoperability in MDD processes. These elements correspond to 1) a set of transformation guidelines to obtain from an i* model an initial version of a model-driven development model; 2) the structural definition of the i* metamodel that is used in the definition of the transformation guidelines involved; and 3) the organizational description of the photography agency, which is the software project used as reference to perform the proof of concepts of our proposal.*

A1.1. Introduction

The success of computer applications increasingly depends on a good understanding of the system requirements. Currently, a requirement specification should include, in addition to software specifications, business models, domain models and other kinds of information that describe the context in which the intended system will operate. During early stages of requirement engineering process, it is necessary to identify and specify how the intended system meets the organizational goals, why the system is needed, what alternatives were considered, what the implications of the alternatives are for the stakeholders, and how the interests and concerns of the stakeholders might be addressed.

Hence, Goal-Oriented Requirements Engineering (GORE) stood out because it is mainly concerned with the stakeholder's intentions and their rationales. Several works on GORE have being proposed: KAOS [179], *i** framework [6], MAPS [15], Non-Functional Requirements (NFR) framework [207]. In all of them, requirement modeling appears to be a core process. However, how to go from requirement models to the corresponding software products is still an open question. To answer this question, we advocate the combined use of GORE and Model-Driven Development (MDD) [1], two complementary model-based approaches.

Thus, it is necessary to use a requirement modeling approach that facilitates the specification of model transformations for the automatic generation of MDD-oriented models. Since present-day technologies (such as ATL or QVT) propose the specification of model transformations driven by metamodels, the use of the *i** approach is a suitable alternative because it has a well-defined syntax [171] and it is possible to find metamodel specifications [7, 172], which can be used as reference for the definition of modeling transformations.

In this appendix, we propose guidelines to generate from an *i** requirement model a conceptual model that is used as input of a MDD process. This MDD process is based on the OO-Method approach [12]. We have chosen OO-Method as a reference MDD technology because it allows the complete generation of the final application from a model definition, and it has been successfully applied to industrial software development by means of the *OlivaNova* tools [64].

Therefore, this work proposes the generation of an initial OO-Method model from an *i** requirement model. This generation is performed by means of a set of transformation guidelines. These transformation guidelines are defined according to an *i** metamodel that has been defined according to the EMOF specification.

To illustrate the transformation guidelines, we have selected a real software project that has been developed in the context of the PROS Research Center [12]: *The Photography Agency System [158]*.

This appendix is organized as follows: Section A1.2 introduces the Photography agency project. Section A1.3 shows the the *i** metamodel that has been defined for the specification of the transformation guidelines. Section A1.4 presents the guidelines to perform the transformation of *i** models into intial OO-Method models. Finally, Section A1.5 shows relevant conclusions obtained.

A1.2. The Photography Agency Project

This section presents the photography agency project that has been developed in the context of the OO-method approach to demonstrate the applicability of the Model-Driven Development, and the automatic model compilation that is performed through the OO-Method industrial technology. We have detailed the development of this software project in the technical report that is presented in [158]. A subset of the organizational description of the photography agency is used to evaluate the proposal for linking *i** modeling and MDD processes. It is important to note that this software project has been defined before to the interoperability scenario related to *i** and OO-Method and before to the interoperability approach presented in this thesis. This is relevant because it assures that the development of the software project has been not affected by the definition of transformation guidelines, verification mechanisms, and metamodels involved in the interoperability approach. Following, the description of the photography agency operation that is considered in thesis is presented.

The photography agency is dedicated to the management of photo reports and their distribution to publishing houses. This photography agency operates with freelance photographers, which must present a request to the production department of the agency. This request contains: the photographer personal information, a description about the owned equipment, and a brief curriculum. An accepted photographer is classified in one of three possible levels for which minimum photography equipment is required. For this, the production department creates a new record for the photographer indicating the level established for the photographer. The possible levels are defined by the commercial department. Furthermore, the commercial department establishes for

each level, the price that will be paid to the photographers and the price that will be charged to the publishing house for each photo.

For the proof of concepts related to the *i** and OO-method linking, we have developed the *i** SR model for the organizational scenario related to the management of work requests. This *i** model is presented in Figure 86.

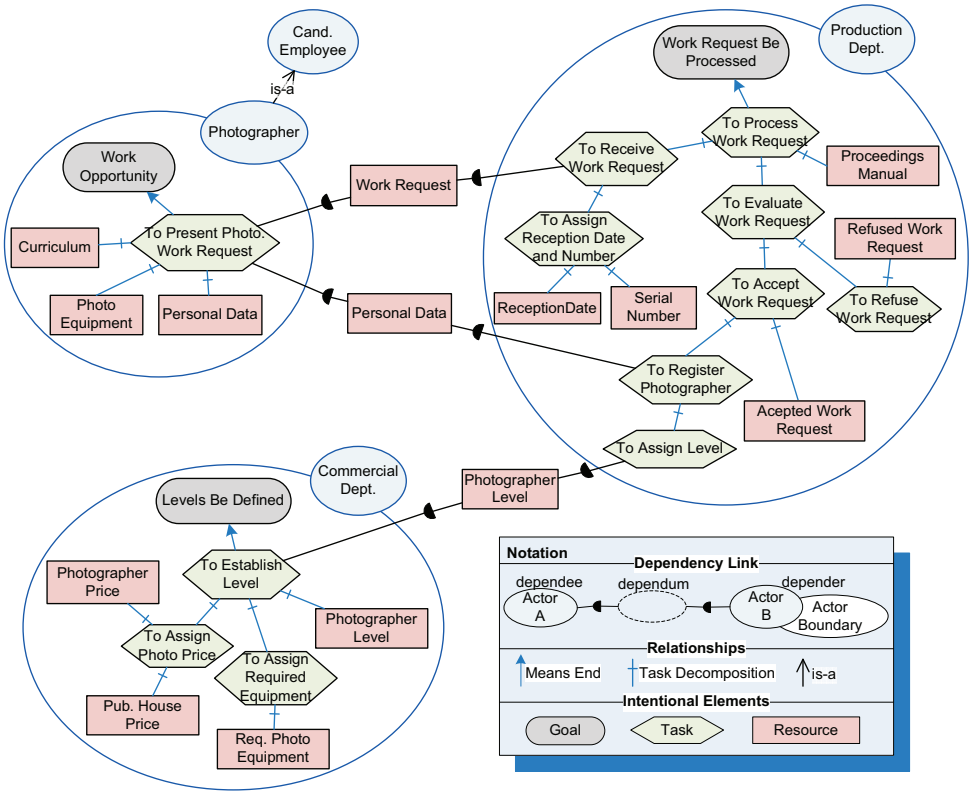

Figure 86. *i** SR model related to photographer work request

The SR model that is presented in Figure 86 captures some of the rationales involved in the processing of a photographer´s work request. For instance, a photographer must present a *Work Request* to the *Production Department* in order to achieve the goal *Work Opportunity*. This is represented by the resource dependency link between the actor *Photographer* and the actor *Production Dept.* To achieve this goal, the photographer must compose a work request that contains: a description of his/her equipment, a brief curriculum, and a book with his/her photographic reports. Finally, this request is processed by the *Production Dep.* actor.

A1.3. The *i** Metamodel

The goal-oriented modeling has proved to be an efficient means of capturing the '*Whys*' and establishing a close relationship with the '*Whats*' [10][16] of the intended systems. GORE is concerned with the use of goals for eliciting, elaborating, structuring, specifying, analyzing, negotiating, documenting, and modifying requirements.

The EMOF metamodel defined for the *i** framework has been developed by taking the proposals presented in [7], [208], [172], and [172] as reference. It also has been considered the specification presented in the *i** guide v3.0 [123], and the metamodel related to the User Requirements Notation (URN) [209], which is a variant of the *i** Framework. The resultant metamodel is presented in Figure 87. This metamodel only considers the structural specification of the *i** constructs, which provide all the necessary information for the application of the interoperability scenarios developed in this thesis and the transformation guidelines presented in this appendix.

For the definition of the presented *i** metamodel, the constructs that are related to the *i** *Strategic Rationale* (SR) model have been considered. The SR model (such as the example *i** model presented in Figure 86) expands the description of a given actor and all the rationale involved on its intentions, providing support for modeling the reasoning of each actor about its intentional relationships. Therefore, the SR model provides additional modeling information (in relation to the SD model), which is useful for the definition of automatic transformation guidelines. The constructs that are considered in the proposed *i** metamodel are defined according to the *i** guide [123] that is published in the official *i** Wiki [171]. In this guide, can also be found the corresponding notation for each *i** construct. Following, the constructs that are involved in the proposed *i** example are presented.

Actor

An actor represents any unit for which intentional dependencies can be ascribed. Actors carry out activities, produce entities, and have desires and needs. Actors are focused on meet their immediate goals and they are concerned about longer-term implications of their structural relationships with other actors, for instance, opportunities and vulnerabilities. Agents, Roles and Positions are sub-units of a complex social actor, each of which is an actor in a more specialized sense.

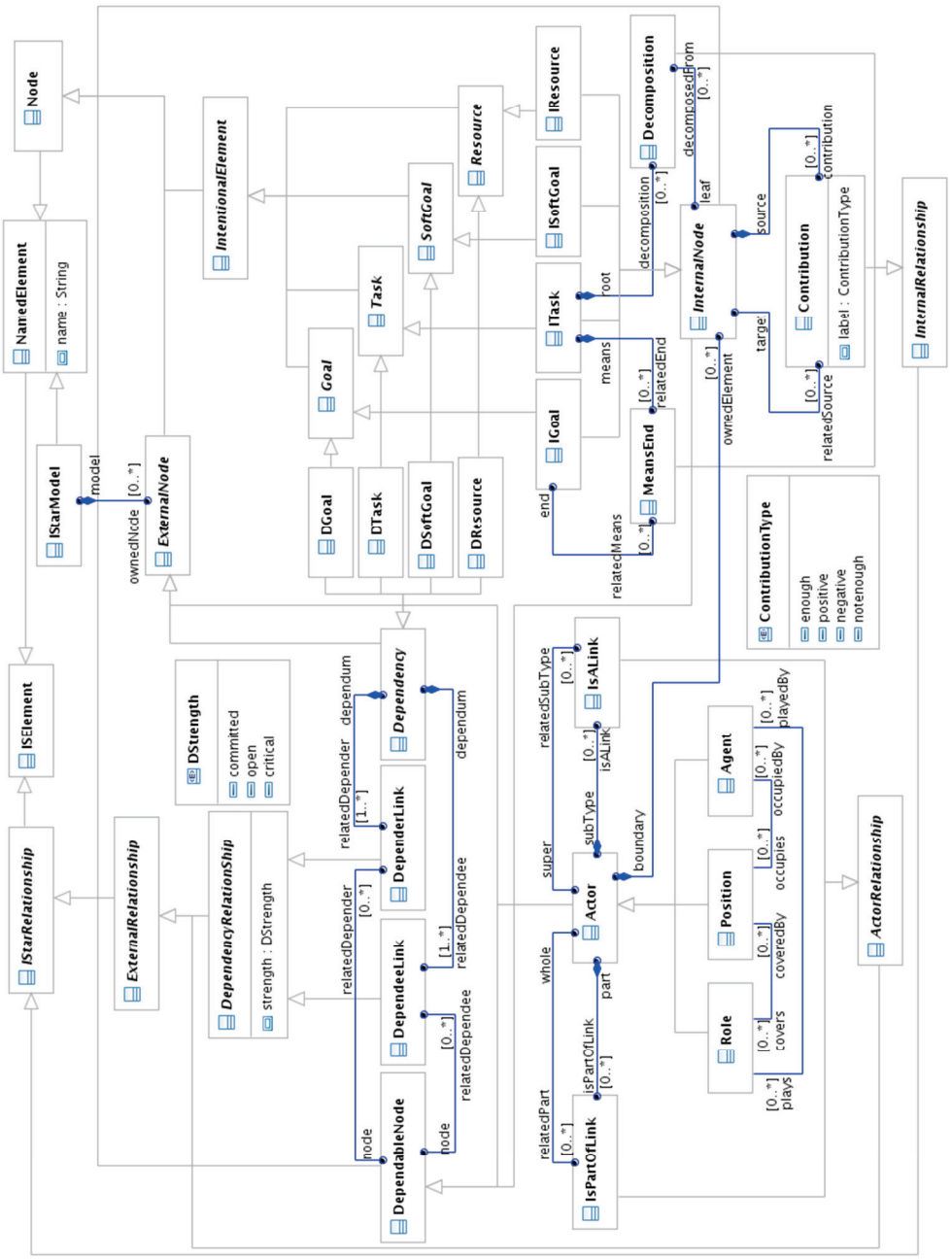

Figure 87. *i** Metamodel

IS-A Association

The IS-A association represents a generalization, with an actor being a specialized case of another actor. This association can be applied between any two instances of the same type of actor.

Dependency Link

A dependency link indicates that exist certain kind of dependency between two actors. The dependency links consist of three elements, the *dependum*, the *dependee* and *depender*. The dependum is the element that generates the dependency between the actors, the *depender* actor is the actor that requires the *dependum,* and the *dependee* is the actor responsible for providing the *dependum* element to the *depender* actor. The dependency links are specialized according to the type of the *dependum*. Thus, it is possible to observe four types of dependency links: *goal*, *resource*, *task*, and *softgoal*.

Goal

A goal in the *i** context is a condition or state of concerns that the actor would like to obtain. It represents an intentional desire of an actor, the specifics of how the goal is to be satisfied is not described by the goal. This can be described through task decomposition.

Resource

A resource is a physical or informational entity that must be available for an actor. This type of element assumes that there are no open issues or questions concerning about how the entity will be produced or provided.

Resource Dependency

In a resource dependency, the *depender* depends on the *dependee* for the availability of an entity (physical or informational). By establishing this dependency, the depender gains the ability to use this entity as a resource. A resource is the final product of some deliberation-action process. In a resource dependency, it is assumed that there are no open issues to be addressed or decisions to be made about the provision or achievement of the Resource entity involved.

Task

A task specifies a particular way of doing something. A more specific description of tasks can be performed by decomposing the tasks involved into further sub-elements.

Means-End Links

These links indicate a relationship between an end, and a means for achieving this end. The "means" is expressed in the form of a task since the notion of task embodies how to do something. The "end" is expressed as a goal.

Task Decomposition

Task decomposition is a link that describes a particular course of action that must be done to perform a certain task. A task element is linked to its component nodes by means of decomposition links. A task can be decomposed into four types of elements: a *subgoal*, a *subtask*, a *resource*, and/or a *softgoal*. These decomposed elements can also be part of dependency links when the reasoning goes beyond of an actor's boundary.

A1.4. The *i** Transformation Process

For the formulation and proper application of the *i** transformation guidelines, we have proposed a specific transformation process to obtain an initial MDD model as output of the *i** model transformation. This process consists in the analysis of the goals defined in the *i** SR model to capture the organizational processes that must be automated in the software system that is required. Then, the intentional elements that are related to these processes are selected (marked). These intentional elements are related to the *i** resources (informational or physical entities) that must be stored in the intended system, and the tasks that must be automated by the system. Finally, the transformation guidelines are applied over the identified elements to obtain an initial conceptual model. This transformation process is graphically depicted in Figure 88 by means of the BPMN [128] notation.

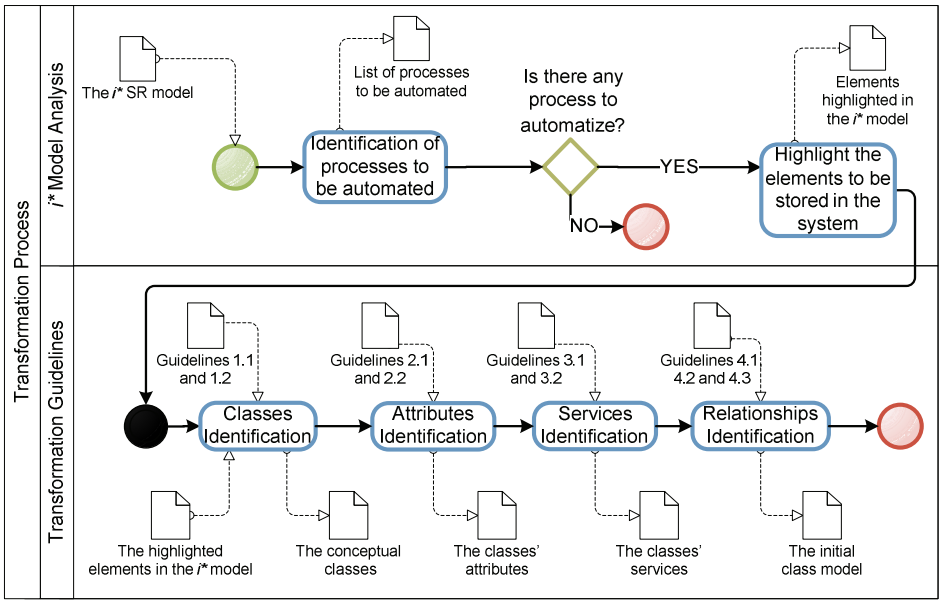

Figure 88. The transformation process modeled with BPMN

A1.4.1. The *i** Model Analysis

According to the transformation process (see Figure 88), this phase is comprised of the following activities: 1) identification of processes to be automated in the intended information system from the *i** SR model; and 2) highlighting of essential elements that are related to the identified process, which must be considered in the software system implementation.

Activity 1: Identification of the process to be automated

In this activity, the goals that are defined in the *i** SR model are analyzed to identify tasks in a means-end links that operationalize these goals. These tasks are the processes to be automatized in the intended system. In the presented *i** SR model, the identified goals are the following: Work opportunity for the Photographer actor; and, A photographer´s work request to be processed by the Production Dep. actor. The tasks that are means to reach these goals are: *To present a work request* and *To process a work request*. The last of these two tasks

is the process that we have decided to automate to exemplify the *i** transformation guidelines.

Activity 2: Highlighting the essential elements

In the second activity related to the *i** model analysis, for each process to be automated, we analyze the respective task-decomposition tree inside the actor boundary. Through this analysis, we select all essential elements that must be considered in the implementation of the intended system. Figure 89 shows the selected (marked) elements for the reference *i** model presented in this annex.

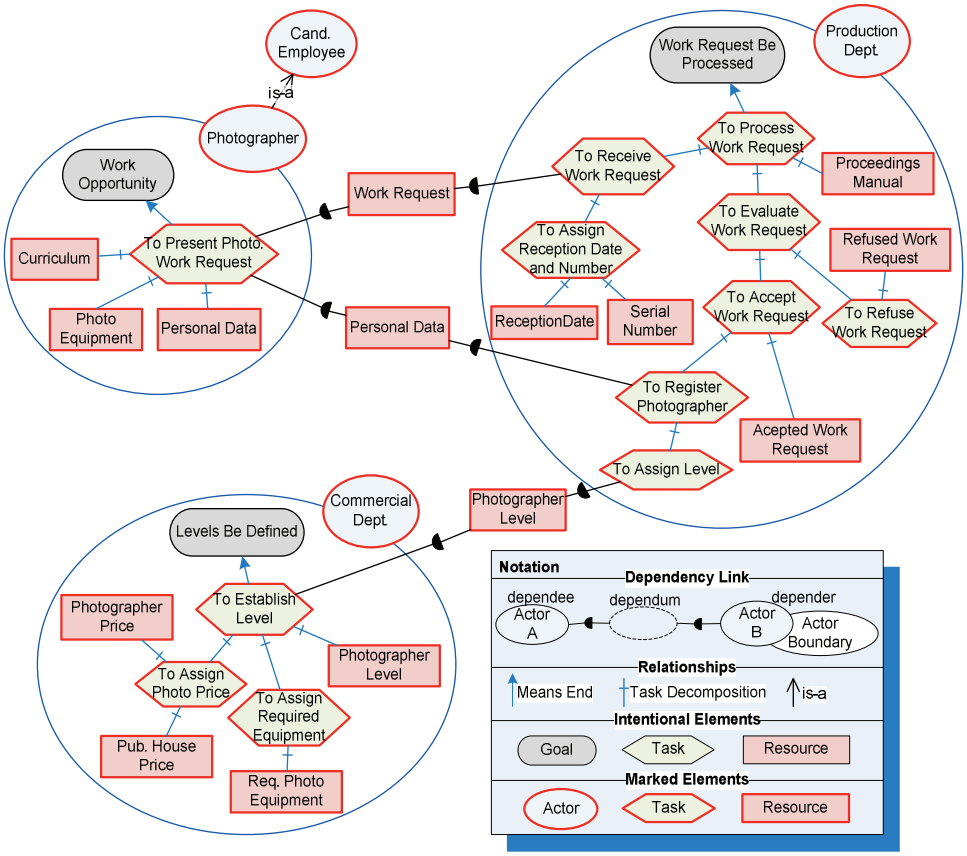

Figure 89. Reference *i** SR model with marked elements to be automated

The selected elements in the *i** SR model are those related with the process to be automated. These selected elements will be translated to elements of the target MDD model using the transformation guidelines presented below. In this

transformation proposal, the target MDD model corresponds to the class model of the OO-Method approach.

A1.4.2. The *i** Transformation Guidelines

In this stage, the guidelines to obtain the corresponding OO-Method class model from an *i** SR model are presented. These guidelines are grouped in four activities Identification of classes, Identification of class attributes, Identification of class services, and Identification of relationships among classes. Figure 90 shows the class model obtained after the application of the transformation guidelines to the reference *i** model.

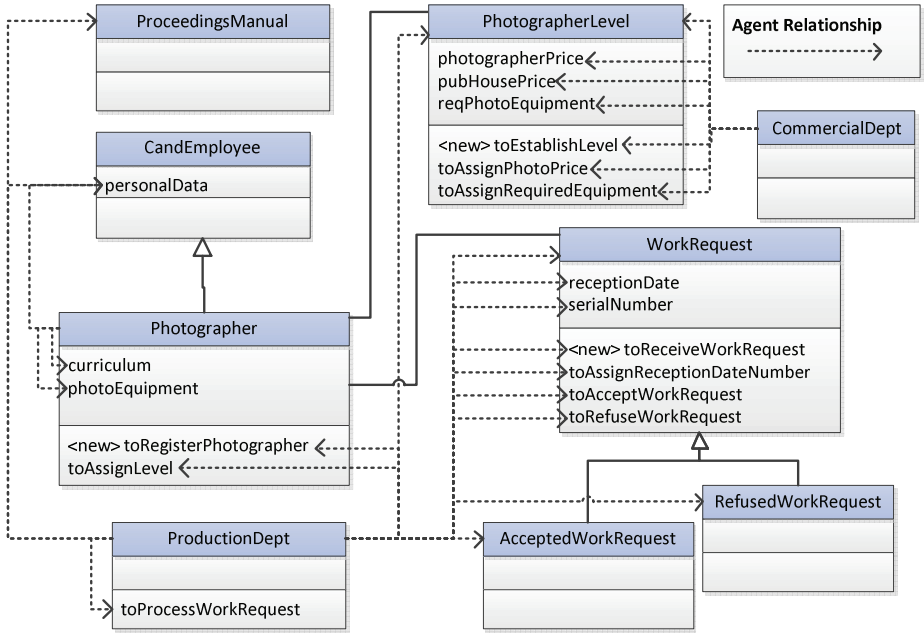

Figure 90. Class model obtained from the application of the proposed guidelines

Activity 1: Identification of classes

This activity deals with the identification of the main classes that should be in the class model. The constructs of the *i** framework that must be considered are the actors and the resource elements that are representing physical entities. The

actors are considered for the generation of classes in the target class model because, by definition [18], an actor is an active entity that carries out actions to achieve goals by using its capabilities. In the case of resources, a resource can represent a physical or informational entity. The physical entities are elements of the real world that have multiple properties. This includes electronic entities such as electronic documents i.e. an electronic invoice. Therefore, the concept of actor and physical entities can be matched to the concept of object in a class model, which implies that both *i** elements are related with the class definition. Thus, for these two *i** constructs, we have obtained the following two transformation guidelines.

Guideline 1.1: An *i** Actor is transformed into a Class of the target class model. The name of the generated Class is obtained from the name of the Actor. In the marked *i** model (see Figure 89) the involved actors are the following:

- Cand. Employee (Candidate Employee)
- Photographer
- Production Dept.
- Commercial Dep.

Guideline 1.2: An *i** Resource that represents a physical entity is transformed into a Class of the target class model. The name of the generated Class is obtained from the name of the Resource. In the marked *i** model (see Figure 89) the involved resources are the following:

- Work Request
- Refused Work Request
- Accepted Work Request
- Proceedings Manual
- Photographer Level

Activity 2: Identification of class attributes

For each class obtained by the application of the first two transformation guidelines, it is necessary to identify those *i** element that can be used to infer attributes of the generated classes. To do this, the resources that represent an

informational entity will be our main target because they represent properties of physical resources or actors. Thus, these informational resources are transformed into attributes of the classes generated from the corresponding actors or physical resources. For the generation of attribute the following transformation guidelines are defined.

Guideline 2.1: If an *i** Resource represents an informational entity related to an actor of the *i** model, then the informational resource is transformed into one attribute of the class generated from the Actor involved. The name of the attribute is obtained from the name of the informational resource. The resources of the reference *i** model (see Figure 89) that are affected by this transformation guideline are the following:

- Personal Data (related to Cand. Employee)
- Curriculum (related to Photographer)
- Photo Equipment (related to Photographer)

Guideline 2.1: If an *i** Resource represents an informational entity (informational resource) related to a resource (physical resource) of the *i** model, then the informational resource is transformed into an attribute of the class generated from the physical resource involved. The name of the attribute is obtained from the name of the informational resource. The resources of the reference *i** model (see Figure 89) that are affected by this transformation guideline are the following:

- Serial Number (related to Work Request)
- Reception Date (related to Work Request)
- Photographer Price (related to Photographer Level)
- Pub. House Price (related to Photographer Level)
- Req. Photo Equipment (related to Photographer Level)

Activity 3: Identification of class services

At this point, the tasks of the *i** SR model and their possible decompositions are inspected (deep search). In the *i** framework, a task specifies a particular way of doing something. When a task is described as a subcomponent of a (higher) task, in a hierarchy of tasks, this restricts the higher task to that particular course of action (a task-decomposition link at the SR model). Moreover, from the practical

experience, a task in the *i** model generally is responsible for achieving a goal and/or for producing a resource. For the identification of class services we consider that a task can be represented by means of a service in the class model. A service of the class model describes a specific behavior of the objects of a class. In the OO-Method approach, a service can be atomic (*Event*) or a composition of other services (*Transaction*). The *i** framework do not provides enough modeling information to determine if a task corresponds to an atomic or composite service, and, hence, the transformation guidelines presented for services generation only produce services as transactions. Additionally, if the task represents a service that groups other services, then this task is transformed into a transaction that is comprised by the grouped transactions. The inferred transactions must be refined later, at design time, to indicate the corresponding events. The services related to creation, deletion, and modification of class instances are created by default when this kind of services cannot be obtained from the application of the transformation guidelines. It is important to remark that the service generation guidelines only consider those tasks that are marked for their automation in the final software system. The guidelines for services identification are the following:

Guideline 3.1: A task that only affects the values of informational resources related to an actor is transformed into a transaction of the class generated from the corresponding actor. The name of generated transaction is inferred from the name of the task. The related resources are defined as arguments of the transaction. If the task is also decomposed in subtasks, then the decomposed subtasks are included in the formula of the transaction generated.

Guideline 3.2: A task that only affects informational resources related to a physical resource is transformed into a transaction of the corresponding physical resource. The name of generated transaction is inferred from the name of the task. The related resources are defined as arguments of the transaction. If the task is also decomposed in subtasks, then the decomposed subtasks are included in the formula of the generated transaction. This guideline is applied to the following tasks of the reference *i** model (see Figure 89):

- To Assign Reception Date and Number (affects *Reception Date* and *Serial Number* from *Work Request*)
- To Assign Photo Price (affects *Pub. House Price* from *Photographer Level*)
- To Assign Required Equipment (affects *Req. Photo Equipment* from *Photographer Level*)

Guideline 3.3: A task that only affects one entity (physical resource or actor) is transformed into a service of the corresponding entity. The name of generated transaction is inferred from the name of the task. If the task is also decomposed in subtasks, then the decomposed subtasks are included in the formula of the generated transaction. In the reference *i** model (see Figure 89), the tasks affected by this guideline are the following:

- To Receive Work Request (affects Work Request)
- To Register Photographer (affects Photographer)
- To Assign Level (affects Photographer)
- To Accept Work Request (affects Work Request)
- To Refuse Work Request (affects Work Request)
- To Establish Level (affects Photographer Level)

Guideline 3.4: A task that affects different physical resources or affects informational resources related to different physical resources is transformed into a transaction of the class generated from the actor that contains such task (according to the corresponding actor boundary). The name of the generated transaction is inferred from the name of the task. The related resources are defined as arguments of the transaction. If the task is also decomposed in subtasks, then the decomposed subtasks are included in the formula of the generated transaction.

Guideline 3.5: A task that does not affect any entity (physical resource or actor) is transformed into a transaction of the class generated from the actor that contains such task (according to the corresponding actor boundary). If the task is also decomposed in subtasks, then the decomposed subtasks are included in the formula of the generated transaction. The name of the generated transaction is inferred from the name of the task. This guideline is applied to the task *To Process Work Request* in the reference *i** model (see Figure 89).

Guideline 3.6: The tasks that participate in a resource dependency, where the *dependum* resource corresponds to a physical resource, are transformed into transactions of the class generated from the *dependum* resource. The names of the generated transactions are inferred from the names of the corresponding tasks. It

is not applicable when the guidelines 3.3 and 3.4 are applied to the involved tasks.

Guideline 3.7: A task that is involved in the generation of an entity (physical resource or actor) is transformed into a creation transaction of the class generated from the corresponding entity. The name of the generated transaction is inferred from the name of the task involved. This guideline is complementary to the guideline 3.3. It is not applicable when the guideline 3.4 is applied to the task involved. This guideline is applied to the following tasks:

- To Receive Work Request (generates Work Request)
- To Register Photographer (affects Photographer)
- To Establish Level (affects Photographer Level)

Activity 4: Identification of relationships among classes

In this activity, the three basics relationships of object-oriented approaches are considered: generalization and associations. However, it is important to remark that *i** mainly focuses on representing strategic concerns by means of intentional elements and their relationships. Therefore, the information of each relationship of the *i** model must be analyzed to derivate the kind of relationships among the generated classes.

The *i** framework offers a concept similar to generalization for the definition of relationship between actors, which is the is-a relationship. This *i** construct has a direct correspondence in the class model. However, for those resource that represent physical entities there is not a clear representation in the *i** framework to determine the occurrence of generalizations. Therefore, it is necessary the expertise of the analyst to determinate when generalizations are present between those physical resources transformed into classes according to the guidelines of *Activity 1*.

The generation of associations among classes is very limited from *i** models, since the *i** framework does not provide enough information to determinate properties like the cardinality of an association. Thus, for the association relationship it must be necessary to add the cardinality information, the role of each association end, and the name of the association at design time. The transformation guidelines related to associations are the following:

Guideline 4.1: If there is a generalization relationship (*is-a* relationship) between two actors of the *i** model that were transformed into classes, then a generalization relationship is defined between the corresponding classes in the class model. In the reference *i** model (see Figure 89), the *is-a* relationship defined between the actors *Cand. Employee* and *Photographer* is affected by this transformation guideline.

Guideline 4.2: If a physical resource of the *i** model is derived from another physical resource, then a generalization relationship is defined from the derived resource to the original resource. This derivation is not explicitly defined in the *i** model, therefore, it is necessary that the analyst indicates the physical resources that hold with this condition. In the reference *i** model (see Figure 89), this situation is present for the resources *Accepted Work Request* and *Refused Work Request*, which are derived from *Work Request*. Thus, two generalization relationships are generated, one between *Accepted Work Request* and *Work Request* and another one between *Refused Work Request* and *Work Request*.

Guideline 4.3: If there exist a resource dependency link where the *dependum*, is transformed into a class, then associations are automatically defined among the class generated from the *dependum* resource and the classes that own the services generated from the tasks involved. If one of the participant tasks is not involved in the transformation process, then the association is defined by considering the actor that owns the corresponding task. In the reference *i** model (see Figure 89), this rule is applied to the resource dependency related to *Work Request* and to the resource dependency related to *Photographer Level*.

Guideline 4.4: For a resource dependency link where the *dependum* is transformed into a class attribute and the *depender* and *dependee* actors are transformed into classes, associations are generated among the class that has the attribute generated from the involved resource and the classes that own the services generated from the tasks involved. In the reference *i** model (see Figure 89), this rule is applied to the resource dependency related to *Personal Data*. However, since the generated services are defined in the class Photographer, no new associations are generated.

The OO-Method approach defines a particular relationship to indicate the visibility that a class has over services or attributes of other classes in the model. This particular relationship is called *agent relationship*, and it is fundamental to

allow the correct execution of the services defined in the class model. The guidelines defined for the agent relationship generation are the following:

Guideline 4.5: For an attribute resulting from the transformation of an informational resource that is defined inside of an actor boundary (internal resource), an agent relationship is created from the class resulting from the transformation of the respective actor (the one that contains the resource) and the attribute related to the informational resource. In the reference *i** model (see Figure 89), the informational resources affected by this guideline are the following:

- Personal Data (related to Photographer boundary)
- Curriculum (related to Photographer boundary)
- Photo Equipment (related to Photographer boundary)
- Serial Number (related to Production Dept. boundary)
- Reception Date (related to Production Dept. boundary)
- Photographer Price (related to Commercial Dept. boundary)
- Pub. House Price (related to Commercial Dept. boundary)
- Req. Photo Equipment (related to Commercial Dept. boundary)

Guideline 4.6: For the services generated form tasks defined inside of an actor boundary, an agent relationship is generated from the class generated from the actor involved and the services generated from the corresponding tasks. Agent relationships are also defined from the class generated from the involved actor and the attributes or classes generated from the resources required by the transformed tasks. In the reference *i** model (see Figure 89), the tasks affected by this guideline are the following:

- To Process Work Request (related to Production Dept. boundary)
- To Receive Work Request (related to Production Dept. boundary)
- To Assign Reception Date and Num. (related to Production Dept. boundary)
- To Accept Work Request (related to Production Dept. boundary)
- To Refuse Work Request (related to Production Dept. boundary)
- To Register Photographer (related to Production Dept. boundary)

- To Assign Level (related to Production Dept. boundary)
- To Establish Level (related to Commercial Dept. boundary)
- To Assign Photo Price (related to Commercial Dept. boundary)
- To Assign Required Equipment (related to Commercial Dept. boundary)

A1.5. Conclusions

In this annex, we have presented the main elements considered to go from *i** requirement models to an initial class model of a MDD approach. In particular, the structural specification of the *i** metamodel considered in this thesis is presented. Furthermore, a set of transformation guidelines are presented to perform the transformation of *i** models (that are based on the defined *i** metamodel) into the corresponding OO-Method class model. These guidelines were systematically designed in accordance with the *i** framework [10]. To illustrate the application of these guidelines, we have manually transformed the presented Photography Agency *i** model into the corresponding OO-Method class model.

Even though the transformation guidelines have been designed in the OO-Method MDD context, the conceptual constructs of the OO-Method class model can be generalized to other object-oriented MDD approaches.

It is important to remark that the class model obtained from the transformation of an *i** model is an incomplete model, which must be refined to obtain a complete representation of the final software system. Thus, the automatic generation of the final software products can be performed by means of a precise model compilation process.

From the application of the transformation guidelines, we have observed that additional modeling information (that is not present in the *i** framework) is necessary to apply the proposed guidelines properly. Therefore, it is not possible to automate the transformation of the *i** models with the current definition of the guidelines and only with the information present in the *i** models. This is the reason why the interoperability approach presented in this thesis has been applied to achieve the automatic linking of *i** models and MDD approaches.

Appendix II.
Doctoral Thesis Development

This doctoral thesis has been developed according to the Design Research methodology. This methodology corresponds to a set of analytical techniques and perspectives to perform research in the information systems area. This appendix introduces the design research methodology and shows its application to the development of this doctoral thesis.

Design research involves the analysis of the use and the performance of designed artifacts to understand, explain and very frequently to improve aspects of information systems. In this context, the design research methodology has been selected for the elaboration and improvement of the artifacts that are required performing the interoperability of modeling approaches in MDD processes, which is the central objective of this doctoral thesis. Next, the main aspects of the design research approach are presented.

A2.1. The Design Research

In a simplified view, research can be considered as an activity that contributes to the understanding of a phenomenon, which is a set of behaviors of some entities that are interesting by a researcher or by a research community [210]. In the design research, the understanding of these phenomena is explained by means of the development of artifacts oriented to satisfy specific functional requirements. Thus, design can be considered as a mapping from function space – a functional requirement constituting a point in this multidimensional space – to attribute space, where an artifact satisfying the mapping constitutes a point in that space [211]. Design then, is knowledge in the form of techniques and methods for performing this mapping – the know-how for implementing an artifact that satisfies a set of functional requirements.

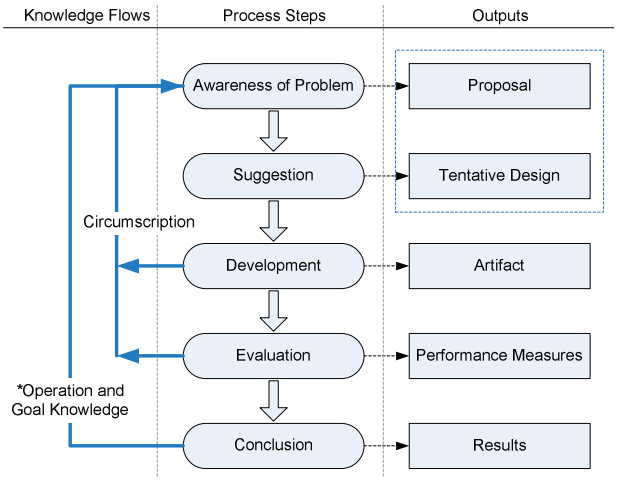

Figure 91. Design research application schema

Figure 91 shows the general schema of the design research application process, by indicating the different outputs that are obtained in each step of the process. The five steps of the action research methodology and the related outputs are explained below (extracted from [210]):

1. **Awareness of Problem:** An awareness of an interesting problem may come from multiple sources: new developments in industry or in a reference discipline. Reading in an allied discipline may also provide the opportunity for application of new findings to the researcher's field. The output of this phase is a Proposal, formal or informal, for a new research effort.
2. **Suggestion:** The Suggestion phase follows immediately behind the proposal and is intimately connected with it as the dotted line around Proposal and Tentative Design (the output of the Suggestion phase) indicates. Moreover, if after consideration of an interesting problem a Tentative Design does not present itself to the researcher; the idea (Proposal) will be set aside. Suggestion is an essentially creative step where new functionality is envisioned based on a novel configuration of either existing or new elements.
3. **Development:** The Tentative Design is implemented in this phase. The techniques for implementation will of course vary depending on the artifact to be constructed. An algorithm may require construction of a formal proof. An expert system embodying novel assumptions about human cognition in an area of interest will require software development, probably using a high-level package or tool. The implementation itself can be very pedestrian and need not involve novelty beyond the state-of-practice for the given artifact; the novelty is primarily in the design, not the construction of the artifact.
4. **Evaluation:** Once constructed, the artifact is evaluated according to criteria that are always implicit and frequently made explicit in the Proposal (Awareness of Problem phase). Deviations from expectations, both quantitative and qualitative are carefully noted and must be tentatively explained. That is, the evaluation phase contains an analytic sub-phase in which hypotheses are made about the behavior of the artifact. The evaluation phase results and additional information gained in the construction and running of the artifact are brought together and fed back to another round of Suggestion (the circumscription arrow of Figure 91). This suggests a new design, frequently preceded by new library research in directions suggested by deviations from theoretical performance.

5. **Conclusion:** This phase is the finale of a specific research effort. Not only are the results of the effort consolidated and "written up" at this phase, but the knowledge gained in the effort is frequently categorized as either "firm" - facts that have been learned and can be repeatedly applied or behavior that can be repeatedly invoked - or as "loose ends" – anomalous behavior that defies explanation and may well serve as the subject of further research.

A2.2. Applying the Design Research

The following outputs were obtained from the application of the design research methodology to the development of this doctoral thesis according to the schema presented in Figure 91.

Step 1: ***Awareness of Problem → Proposal.*** In the context of our research, we have found that a proposal for the interoperability among different modeling approaches in a MDD environment does not exist yet. Therefore, in this thesis we propose a solution that allows this interoperability possible by means of the automatic generation of the required metamodels extension for customization of the modeling languages and the transparent interchange of the models involved.

Step 2: ***Suggestion → Tentative Design.*** For the elaboration of an appropriate MDD interoperability approach is necessary to obtain a solution aligned with the proposal defined. In this process we must consider different artifacts related to the correct interoperability, these artifacts are:

- A mechanism that allows the identification of equivalences among modeling approaches to interoperate.
- A mechanism that allows the differences among modeling languages to be managed to prevent the loss of information during the interchange of models.
- A mechanism for the automatic generation of the modeling languages extensions that solve the differences among modeling languages.
- A mechanism for automatic interchange of models that allows the automatic modeling languages interoperability to be obtained.

For the specification of the modeling languages, the definition of metamodels is used by taking as reference the MOF standard [38]. Figure 92 shows the tentative design of the interoperability process by indicating the different

artifacts involved. The basis for the elaboration of this tentative design has been presented in [212].

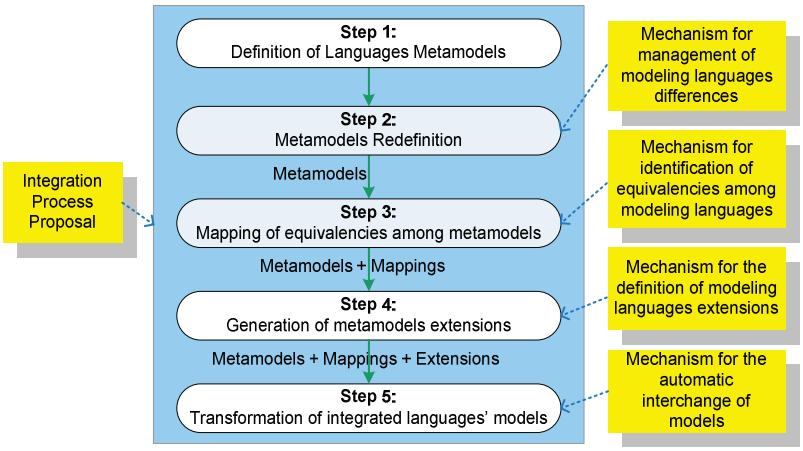

Figure 92. Interoperability Process – Tentative Design

Step 3: ***Development → Artifacts.*** With the development of the artifacts defined in the tentative design presented above, a first version of our interoperability proposal is obtained (see [18]). For the implementation of these artifacts, different aspects has been considered such as definition of UML profiles from DSML metamodels [17, 19, 51, 54], correct use of metamodels in software engineering [129], UML profile implementations [43], interchange between modeling approaches [130, 136], implementation of model interchange tools [132], and new UML profile features [39].

The artifacts obtained were the following:

- **MDD Interoperability Process.** This process integrates different modeling languages by means of metamodels extensions implemented through an automatically generated UML profile and models transformation that are driven by the interchange information obtained during the UML profile generation. Thus, it is possible to observe that the automatic generation of the required metamodel extensions is the core of the interoperability process.
- **Integration Metamodel.** The integration metamodel is the solution developed to provide a mechanism for managing the differences between the modeling languages that will be integrated, and to identify the equivalences between these modeling languages. The initial specification of the Integration Metamodel and the systematic approach proposed for its definition have been

presented in [127]. However, this initial specification has been improved throughout the development of this doctoral thesis to obtain a better integration solution for multiple modeling languages.

- **Automatic UML Profile Generation.** This artifact corresponds to the mechanism implemented for the definition of modeling languages extensions. This is a two steps process that automatically generates a UML profile from an Integration Metamodel (see section 3.1). The two steps that comprise the UML profile generation are:

1. **Metamodels Comparison:** Identifies the differences between an Integration Metamodel and the metamodel of the modeling language to be customized to obtain the metamodels extensions that must be implemented.
2. **Integration Metamodel Transformation:** A set of transformation rules that are defined to automatically transform the Integration Metamodel in the UML profile that implements required metamodel extensions.

- **Models interchange approach.** This approach provides the transformation mechanisms for the interchange of models that are defined with the integrated modeling languages. The interchange approach is based on the information obtained during the UML profile generation, and a set of transformation rules [18].

Step 4: ***Evaluation → Performance Measures:*** The evaluation of the integration process and the rest of the developed related artifacts is performed by means of the linking of *i** framework and UML with the OO-Method MDD development process.

UML has been chosen due to the relevance of this modeling language in the information science community and because UML provides a standard customization mechanism to adapt its syntax to specific domains [33]. In particular, we have considered the use of the UML class model, which is the most used UML model [213]. The UML class model is very close to class model of the MDD approach considered in this thesis (the OO-method class model), and both modeling approaches are at the same abstraction level. In fact, the OO-Method class model could be considered as a subset of the UML class model that introduces new properties for the automatic model compilation process. This prevents the complexity that may arise in the integration of modeling approaches of different domains and abstraction levels. Therefore, the use of UML class model is relevant to obtain an initial approach for modeling languages integration.

The *i** framework [6] has been chosen since it is one of the most widespread modeling and reasoning frameworks [162-164] and it is also well documented [168]. The *i** framework is framed in the context of *Goal-Oriented Requirement Engineering* (GORE) [165]. Thus, we want to tackle the linking of requirement modeling with existing MDD processes. The use of *i** introduces a new complexity aspect to the improvement of the interoperability approach, which is, the linking of modeling languages at different abstraction levels [166]; *i** models are at analysis level while most of the MDD-oriented models, such as the OO-Method conceptual model, are at design level. Additionally, at difference of UML and OO-method class models, the constructs provided by the *i** framework are quite different than the constructs of class models, which demands that the application framework for the interoperability approach be improved to support these differences.

Figure 93 shows the general integration schema for the linking of these three modeling languages (*i**, UML, and OO-Method) by using the interoperability process proposed in thesis. Thus, it is possible to achieve the automatic interoperability of *i** and UML in the OO-Method MDD process.

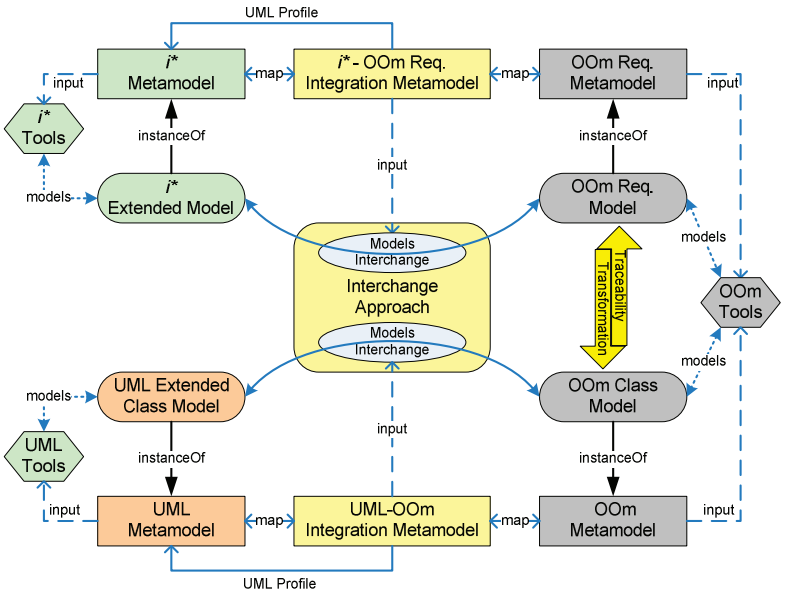

Figure 93. Schema for the integration of *i**, OO-Method, and UML.

The results obtained from the application of the proposed integration approach have provided valuable information for the improvement of the

interoperability in MDD processes. The capability of improving interoperability approach by means of the application the interoperability artifacts defined is one of the main aspects that we have considered for the selection of the design research methodology. This corresponds to the circumscription arrow presented in Figure 91. In addition, the interoperability scenarios developed in the context of an industrial MDD process have provided relevant information to answers many open questions in relation to: the automatic linking of requirement models in MDD environments [32], the definition of a complete process to automate the generation of modeling languages extensions [127, 135], the automatic interchange of models to support different modeling perspectives [136], and the application of current modeling technologies to obtain automatic software products by means of automatic compilation processes [18, 138, 212].

Step 5: *Conclusion → Results.* Finally, after the evaluation of the developed artifacts and the improvement of the initial interoperability approach (obtained in the tentative design), the resultant integration solution proposed in this doctoral thesis is obtained. From the evolving nature of the selected research methodology, further research lines for the development of new artifacts, new interoperability objectives, and further improvements are obtained.