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

Chapter I. Introduction 1

1.1. Motivation 4
1.2. Problem Statement 5
1.3. Objectives 7
1.4. Thesis Development Context 9
1.5. Thesis Structure 10

Chapter II. Background 13

2.1. Modeling Language Customization 14
2.2. The OO-Method MDD Approach 24
2.3. The i*Framework 29

Chapter III. Related Work 33

3.1. Planning the Systematic Review 34
3.2. Revision of Interoperability Approaches 38
3.3. A Common Interoperability Framework 46
3.4. Conclusions 48

Chapter IV. Achieving MDD Interoperability 49

4.1. An MDD Interoperability Model 50
4.2. Challenges for Integration of Modeling Languages 57
4.3. The MDD Interoperability Process 60
4.4. Conclusions 63

Chapter V. The Integration Metamodel 65

5.1. Background 68
5.2. Improving the Integration of Modeling Languages 69
5.3. Systematic Definition of an Integration Metamodel 73
5.4. Implementing the Integration Metamodel 78
5.5. Benefits of the Integration Metamodel 79
5.6. Conclusions 82

Chapter VI. Automatic UML Profile Generation 85

6.1. Background 87
6.2. Identification of Metamodel Extensions 89
6.3. Integration Metamodel Transformation 95
6.4. Applying the Transformation Rules 107

6.5. Conclusions 109

Chapter VII. Generation of Model Interchange Mechanisms 111

7.1. Background 113
7.2. Model Interchange Proposal 115
7.3. Applying the Interchange Proposal 119
7.4. Comparing the Model Interchange Approach 126
7.5. Conclusions 127

Chapter VIII. Linking UML and MDD Approaches 131

8.1. Introduction 132
8.2. The applicability of the UML association in MDD Processes 133
8.3. The Semantics Proposed to Customize the UML Association 135
8.4. Integration of the Proposed Semantics into UML 141
8.5. Compiling the Extended UML Association 149
8.6. Conclusions 153

Chapter IX. Linking Goal-Oriented Modeling and Model-Driven Development 155

9.1. Introduction 156
9.2. Background 157
9.3. Applying the Interoperability Approach to *i** and OO-Method 160
9.4. Analysis of the Proposal and Discussion 171
9.5. Conclusion 173

Chapter X. Automatic Verification of Models for MDD Interoperability 175

10.1. Introduction 176
10.2. Background 177
10.3. Integration of Verification Measures into the *i** Framework 182
10.4. Applying the *i** Verification Measures 192
10.5. Evaluating the Verification Approach 198
10.6. Overall Analysis 210
10.7. Conclusions 211

Chapter XI. Conclusions 213

11.1. Thesis Contributions 215
11.2. Future work 219
11.3. Publications 221

References 225

Appendix I. Transformation of *i** Models into MDD-Oriented Models 245

A1.1. INTRODUCTION 246
A1.2. THE PHOTOGRAPHY AGENCY PROJECT 247
A1.3. THE *i** METAMODEL 249
A1.4. THE *i** TRANSFORMATION PROCESS 252
A1.5. CONCLUSIONS 263

Appendix II. Doctoral Thesis Development 265

A2.1. THE DESIGN RESEARCH 266
A2.2. APPLYING THE DESIGN RESEARCH 268

List of Figures

Figure 1. The MDA schema 3
Figure 2. UML profile example 16
Figure 3. Property redefinition example 18
Figure 4. The OO-Method Software Production Process 25
Figure 5. Compilation alternatives provided by the Olivanova technology 26
Figure 6. The *i** notation 29
Figure 7. Strategic Dependency model of a buyer-driven e-commerce system 30
Figure 8. Strategic Rationale model of a buyer-driven e-commerce system 32
Figure 9. MDD-oriented Interoperability Framework 47
Figure 10. LCIM Model 52
Figure 11. MDD Interoperability Model 54
Figure 12. MDD Interoperability Model Instantiated 56
Figure 13. MDD Interoperability Process 61
Figure 14. DSML metamodel example 70
Figure 15. Integration Metamodel related to a binary association between classes 72
Figure 16. Systematic Approach for Integration Metamodel Definition 75
Figure 17. Fixing mapping problems for Integration Metamodel definition 75
Figure 18. Example of a Integration Metamodel transformation 80
Figure 19. Automatic UML Profile Generation Process 89
Figure 20. Example of an equivalent association without type difference 90
Figure 21. Example of an equivalent association with type difference 91
Figure 22. Example of mapping of equivalent attributes 91
Figure 23. Example of enumeration mappings 92
Figure 24. Example of extension identification from new attributes and cardinality differences 93
Figure 25. Metamodel Comparison Results 94
Figure 26. Generic transformation example for Rule 1 96
Figure 27. Mapping obtained for the example related to Rule 1 97
Figure 28. Generic example for the transformation rules 2 to 4 98
Figure 29. Mapping obtained for the transformation example related to rules 2 to 4 99
Figure 30. Generic example for transformation rule 5 100
Figure 31. Mapping obtained for the transformation example related to Rule 5 100
Figure 32. Generic example for transformation rule 6 101
Figure 33. Mapping obtained for the transformation example related to Rule 6 102

Figure 34. Generic example for transformation rules 7, 8 and 9. 103
Figure 35. Mapping obtained for the transformation example related to rules 7 to 9 104
Figure 36. Generic example for Rule 10 105
Figure 37. Generic example for Rule 11 107
Figure 38. Mapping obtained for the example related to Rule 11 107
Figure 39. UML profile generated for the example Integration Metamodel 108
Figure 40. Mapping generated for the UML profile presented in Figure 39 109
Figure 41. Schema of the Modeling Languages Interchange Proposal 117
Figure 42. Interchange proposal application example 118
Figure 43. Schema for the implementation of the interchange proposal in OO-Method and UML 121
Figure 44. UML model extended with the OO-Method UML profile 122
Figure 45. XMI importer application example 123
Figure 46. OO-Method presentation model 124
Figure 47. Generated application 124
Figure 48. Functional size report generated for the application generated from the example UML model 125
Figure 49. Example UML model 136
Figure 50. Example of Identification Function in the association 137
Figure 51. Example of association temporality 139
Figure 52. : The Integration Metamodel of the proposed association semantics 143
Figure 53. UML Profile generated from the defined Integration Metamodel 148
Figure 54. Example UML model extended with the generated UML profile 149
Figure 55. Compilation alternatives provided by the Olivanova technology 150
Figure 56. Diagram of the SQL database generated from the example UML model 151
Figure 57. Execution of the service that creates new instances of the class Reservation 152
Figure 58. : Execution of the service *del_association* with a precondition 153
Figure 59. Basic goal-oriented requirements and MDD linking schema 159
Figure 60. *i** example model 160
Figure 61. The *i** metamodel for the example model 161
Figure 62. The OO-Method metamodel for the linking example 162
Figure 63. OO-Method requirement metamodel for the *i** linking example 164
Figure 64. General Schema of the proposed linking process 166
Figure 65. Interoperability proposal applied to *i** and OO-Method 166
Figure 66. Integration Metamodel for the integration example 167
Figure 67. UML Profile generated from the Integration Metamodel of the example 169
Figure 68. Extended example *i** model and the OO-Method class model generated 170
Figure 69. Transformations to obtain an OO-Method class model from an *i** model 171

Figure 70. Example *i** SR Model. 180
Figure 71. Process for definition of *i** verification measures 183
Figure 72. Application of the GQM approach. 183
Figure 73. EMOF *i** Metamodel 188
Figure 74. Verification Model and Mapping Information 189
Figure 75. UML Profile to extend the *i** metamodel with the verification measures 191
Figure 76. Process for the application of verification measures 192
Figure 77. Example *i** Model extended with the generated UML Profile 193
Figure 78. Class model generated from the example *i** model 194
Figure 79. Improved *i** model 196
Figure 80. Class model generated from the improved *i** model 197
Figure 81. Second *i** Model (ISTAR2) for Photography Agency Description 199
Figure 82. Initial class model generated from ISTAR2 without improvements 201
Figure 83. Experimental tasks 202
Figure 84. Second *i** model (ISTAR2) improved with the verification measure results 207
Figure 85. Class model obtained from the improved version of the second *i** model 208
Figure 86. *i** SR model related to photographer work request 248
Figure 87. *i** Metamodel 250
Figure 88. The transformation process modeled with BPMN 253
Figure 89. Reference *i** SR model with marked elements to be automated 254
Figure 90. Class model obtained from the application of the proposed guidelines 255
Figure 91. Design research application schema 266
Figure 92. Interoperability Process – Tentative Design 269
Figure 93. Schema for the integration of *i**, OO-Method, and UML 271

List of Tables

Table 1. Summary of the studies analyzed in the systematic review 45
Table 2. The LISI reference model 51
Table 3. Mapping obtained from the Integration Metamodel generation 77
Table 4. Metamodel Extensions Identified 95
Table 5. OCL constraints for association ends 144
Table 6. : OCL constraints for shared events 145
Table 7. Equivalences between the Integration Metamodel and the UML Metamodel 146
Table 8. Comparison between the Integration Metamodel and the UML Metamodel 147
Table 9. Guidelines for transformation of *i** models into OO-Method Class Models 163
Table 10. Integration Metamodel and the *i** metamodel mapping 168
Table 11. OO-Method requirements metamodel and Integration Metamodel mappings 169
Table 12. Guidelines for the transformation of *i** models into OO-Method class models 181
Table 13. Characteristics of measure Wrong Attribute Generation (WAG) 185
Table 14. Characteristics of measure Wrong Service Generation (WSG) 185
Table 15. Characteristics of measure Non-Accessible Element (NAE) 186
Table 16. Characteristics of measure Non-Instantiable Class (NIC) 187
Table 17. WAG measure specification in the OCL language 190
Table 18. Results obtained from measures evaluation. 193
Table 19. Tagged values related to the example *i** Model 194
Table 20. Fixing guidelines related to the verification measures 195
Table 21. Tagged values changed in the improved *i** Model 197
Table 22. Tagged values related to the Second *i** Model for Photography Agency 200
Table 23. First generation of the MDD models. 204
Table 24. Application of the verification measures to ISTAR1 and ISTAR2 205
Table 25. Tagged values changed in the improved *i** Model 208
Table 26. Second generation of the MDD models. 208
Table 27. Experiment results 209
Table 28. Publication Summary 223

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.

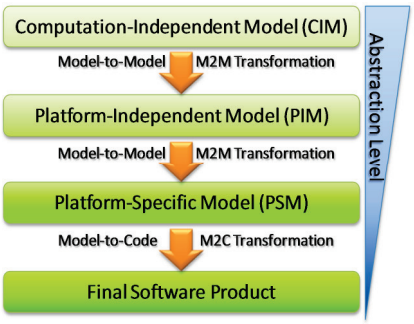

Figure 1. The MDA schema

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.

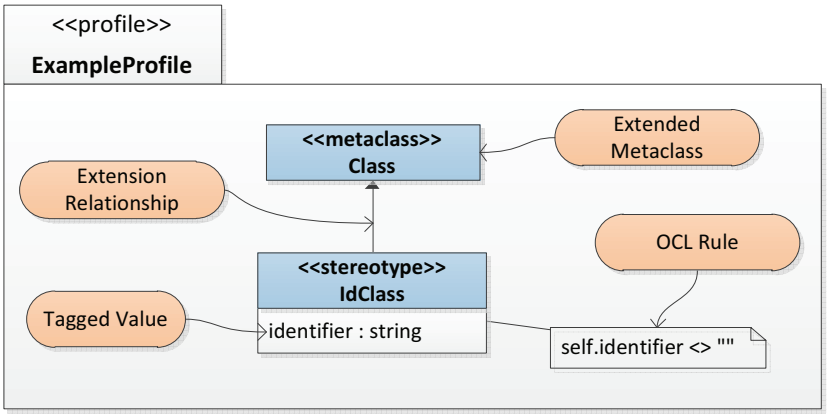

Figure 2. UML profile example

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.

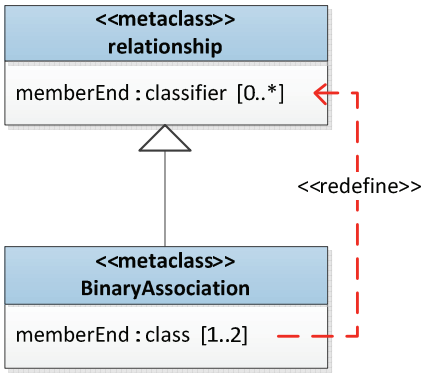

Figure 3. Property redefinition example

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.

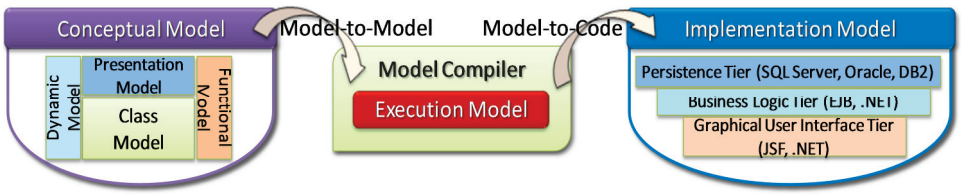

Figure 4. The OO-Method Software Production Process

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.

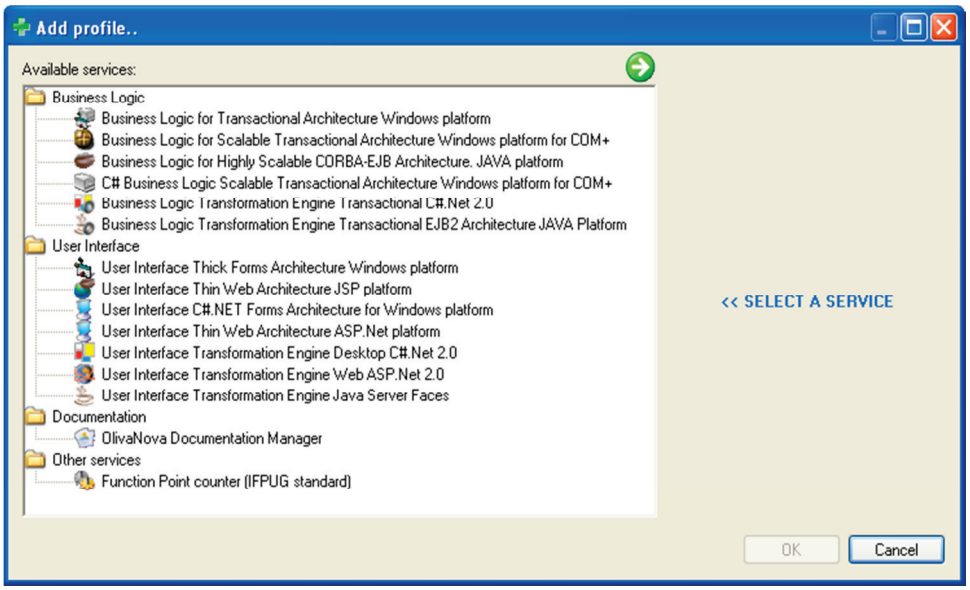

Figure 5. Compilation alternatives provided by the Olivanova technology

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