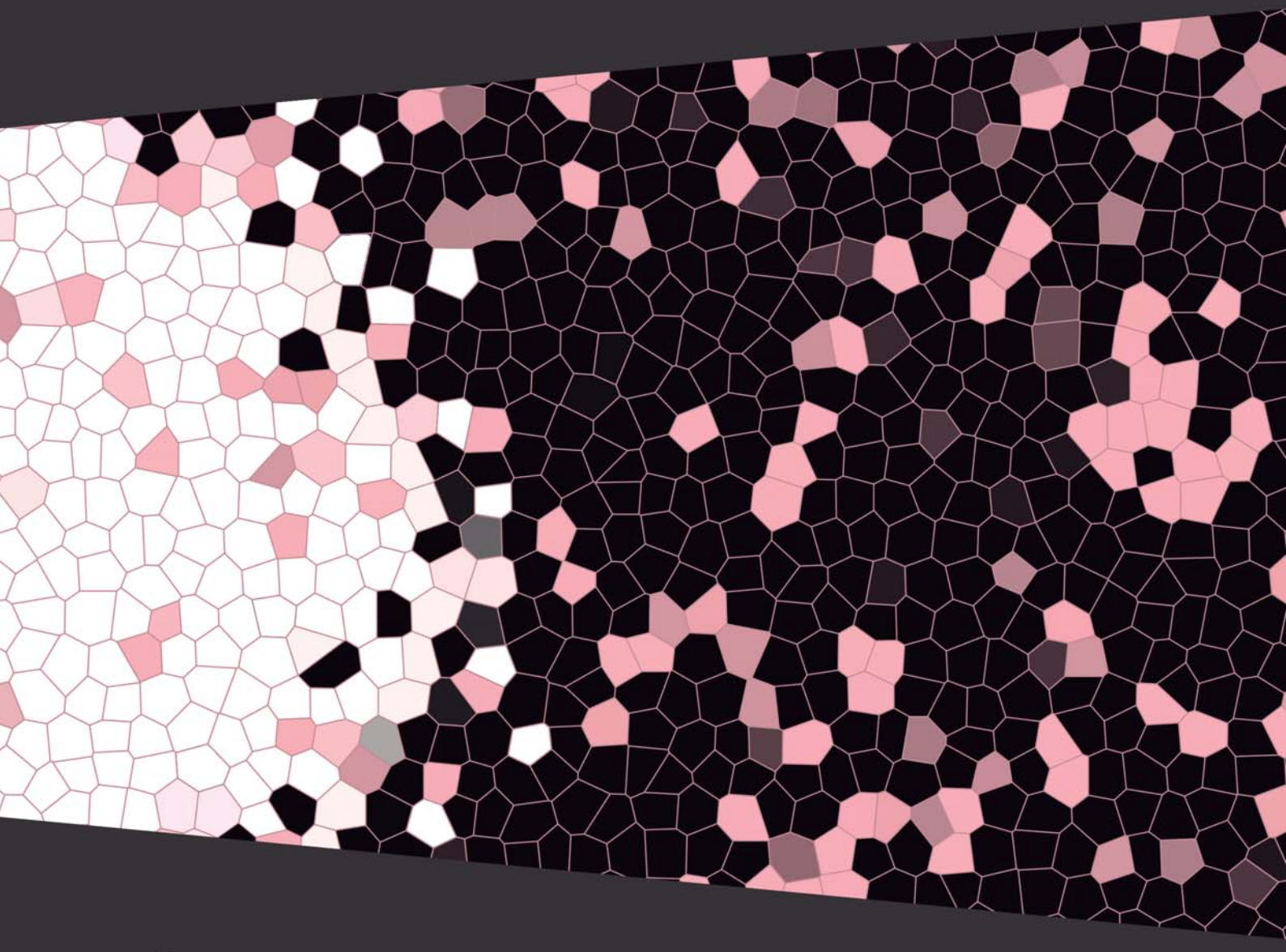
Achieving Non-intrusive Interoperability between Models for Involving Users in Modeling Tasks

María Francisca Pérez Pérez

November 2015



UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Thesis Supervisors:
Pedro José Valderas Aranda
Joan Fons i Cors

Doctoral Thesis

Achieving Non-intrusive Interoperability between Models for Involving Users in Modeling Tasks

María Francisca Pérez Pérez

November 2015

Supervisors:

Dr. Pedro José Valderas Aranda

Dr. Joan Fons i Cors

UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Achieving Non-intrusive Interoperability between Models for Involving Users in Modeling Tasks

This report was prepared by

María Francisca Pérez Pérez

Supervisors:

Dr. Pedro José Valderas Aranda (Universitat Politècnica de València)

Dr. Joan Fons i Cors (Universitat Politècnica de València)

Dissertation Committee:

Dr. Jordi Cabot Sagrera (INRIA - École des Mines de Nantes)

Dr. Antonio Ruiz Cortés (Universidad de Sevilla)

Dr. Vicente Pelechano Ferragud (Universitat Politècnica de València)

ACKNOWLEDGEMENTS

Many people have played in a way or another an important role during these years of hard work to reach the completion of this thesis journey. Firstly, I would like to express my sincere gratitude to my thesis advisors, Dr. Pedro Valderas and Dr. Joan Fons, for their patience, advice, and insightful comments. Their guidance helped me in all the time of research and writing of this thesis.

Furthermore, I am grateful to Dr. Vicente Pelechano for enlightening me the first glance of research and gave me access to the research group with its research facilities. Without his support it would not be possible to conduct this research. I also want to express my gratitude to Prof. Óscar Pastor for his advices, for always having a smile for me, and for showing me the funniest side and challenges of being the header of a big research center as the ProS is.

Also, I would like to thank the external members of my thesis committee, Dr. Jordi Cabot and Dr. Antonio Ruiz for taking time out from their busy schedules and for their valuable comments. In addition to the committee in charge of evaluating this thesis and my two thesis advisors, this thesis was reviewed by two external reviewers, Dr. Francisco José García and Dr. Luis Olsina, so I also want to thank their comments and suggestions.

I also owe a great deal of gratitude to a number of other friends and colleagues at the ProS - both past and present. I have learned a great deal from all of them and they gave me the opportunity to know different cultures

and perspectives. Those especially deserving of a mention include Dr. Carlos Cetina, Dr. Giovanni Giachetti, Dr. Pau Giner, Dr. Beatriz Marín, and Dr. Estefanía Serral, who were my lab-mates. Thanks for the fun, support and the good times we spent inside and outside the lab. Lastly, I must also show my gratitude to the *Universidad San Jorge* and all my colleagues of the *Escuela Politécnica Superior* for their encouragement to finish this thesis. I feel privileged to work there because I am surrounded by motivated and marvelous people.

I want to thank my parents, Kathy and Ángel, for bringing me into this world and for supporting me unconditionally in the pursuits of my life. Thanks Dad for showing me the amazing world of computers and other electronic devices since I saw you work with them in my childhood. Thanks Mom for being there whenever and whatever I need since I was born. And, of course, thank you both for your love and constant support in all maters of life. Your confidence in me has enhanced my ability to get through it all.

I would also like to express my thanks to the rest of the family and friends for their constant support, encouragement, and comprehensiveness even the distance and the hard work make me miss some important events and enjoy their company less than I would. I wish to sincerely thank Marcela Ruiz. Marcela is an inexhaustible source of emotional support and the best fun stuff. I feel very fortunate that thanks to this thesis journey I had the chance to meet her, learn a lot from her, and share exciting adventures.

Last but not the least, I would like to thank my partner in life Carlos. I have been so lucky having him during this journey. His love, bullet-proof patience, support, and encourage is giving me all that I need, and more happiness and experiences than I could have ever imagined.

ABSTRACT

Model-Driven Development (MDD) promotes models as the cornerstone in the software development process, thereby displacing source code as the development process's main feature. Although this model-centric schema claims advantages over traditional software development (e.g., the code could be automatically generated from the models), it does not have the level of adoption that has been expected.

The literature review reveals a broad agreement in the fact that end-users may develop and adapt systems themselves but the complexity in modeling standards and the lack of modeling skills prevents their active involvement in modeling tasks of existing MDD processes. To overcome this, end-users should be provided with different modeling languages that use concepts, which fit their particular skills, context and needs.

This challenge is the main goal of this thesis, which is addressed by combining the End-user Development and the Model-Driven Development fields. This work starts with the involvement of end-users into the modeling tasks using a tool-supported visual modeling language that allows end-users to select and customize system features of pervasive systems using closer concepts for them. Afterwards, this thesis shows the necessity of enriching existing MDD processes for supporting the development of a new generation of software systems (e.g., smart health) that require expertise in a variety of domains. Consequently, different types of users (e.g., scientists, engineers and end-users) must actively participate in the description of model fragments

that depend on their expertise using a different modeling language. Thus, users are able to collaborate to obtain a unified system description. At this point, it becomes necessary to provide mechanisms that transforms models fragments from one modeling language to another, delimits which model fragments are described by a different user, and integrates those model fragments.

To provide this, the presented approach encompasses variability management in a novel way to enable collaborative modeling by supporting both the selection of model fragments of the system that may be described using a different modeling language, and the integration of those model fragments once they are described. Furthermore, interoperability mechanisms bridge two different modeling languages in a non-intrusive way with the structure of models by transforming the description of gaps. Thus, our proposal could enrich models of existing MDD processes with model fragments that have been described using a different modeling language, which could make users feel confident to adopt models for describing domain-specific content and could help to adopt MDD processes.

The proposal has been validated in three case studies from different levels of complexity and domains: smart home systems, web information systems, and biomechanical protocols. The results have proven the applicability and feasibility of our approach to actively involve different types of users (end-users with software professionals, domain experts with software development experts, and doctors with biomedical engineers, respectively) in model descriptions of existing MDD processes using a different modeling language.

RESUMEN

En el Desarrollo de Software Dirigido por Modelos (DSDM) los modelos son la piedra angular del proceso de desarrollo de software, desplazando así al código fuente como artefacto principal. Aunque este enfoque centrado en modelos ofrece ventajas sobre el desarrollo de software tradicional (por ejemplo, la generación de código de forma automática a partir de los modelos) no tiene el nivel de adopción esperado.

La literatura científica revela un amplio acuerdo en el hecho de que los usuarios finales puedan ellos mismos desarrollar y adaptar los sistemas pero la complejidad de los estándares de modelado y la carencia de habilidades de modelado impide su participación activa en procesos DSDM existentes. Para lograrlo, los usuarios finales deben disponer de lenguajes de modelado diferentes con conceptos adaptados a sus habilidades, contexto y necesidades.

Este desafío es el objetivo principal de esta tesis que se aborda combinando las ideas del desarrollo orientado al usuario final y el DSDM. Este trabajo comienza involucrando usuarios finales en tareas de modelado con una herramienta que les proporciona un lenguaje de modelado visual para seleccionar y personalizar características de un sistema pervasivo utilizando conceptos familiares para ellos. Después, esta tesis motiva la necesidad de enriquecer procesos de DSDM existentes para soportar el desarrollo de una nueva generación de sistemas software (por ejemplo, salud inteligente) que requieren conocimientos especializados en una variedad de dominios. Consecuentemente, diferentes tipos de usuarios (por ejemplo, científicos,

ingenieros y usuarios finales) deben participar activamente en la descripción de fragmentos de modelos que dependen de su experiencia utilizando un lenguaje de modelado diferente. De este modo, los usuarios pueden colaborar para obtener una descripción del sistema unificada. En este punto, es necesario proporcionar mecanismos que transformen e integren los fragmentos de un lenguaje de modelado a otro y delimiten qué fragmentos se describen por un usuario diferente.

Para proporcionar esto, la propuesta presentada utiliza la gestión de variabilidad de forma novedosa para permitir modelado colaborativo seleccionando fragmentos de un modelo del sistema que pueden ser descritos utilizando un lenguaje de modelado diferente y, la integración de esos fragmentos una vez que hayan sido descritos. Además, la propuesta utiliza mecanismos de interoperabilidad para conectar dos lenguajes de modelado diferentes transformando la descripción de los fragmentos de una manera no invasiva con su estructura. Por tanto, nuestra propuesta puede enriquecer los modelos de procesos DSDM existentes con fragmentos de modelos que han sido descritos con un lenguaje diferente y esto, podría hacer que los usuarios se sientan seguros al adoptar modelos para describir contenido de dominio específico y podría ayudar a adoptar procesos DSDM.

La propuesta ha sido validada en tres casos de estudio con diferentes niveles de complejidad y dominios: sistemas para el hogar inteligente, sistemas de información web y protocolos biomecánicos. Los resultados han demostrado la aplicabilidad y viabilidad de nuestra propuesta para involucrar diferentes tipos de usuarios (usuarios finales con profesionales de software, expertos en el dominio con expertos en desarrollo de software y, médicos con ingenieros biomédicos, respectivamente) en descripciones de modelos de procesos DSDM existentes utilizando un lenguaje de modelado diferente.

RESUM

En el Desenvolupament de Programari Dirigit per Models (DPDM) els models són la pedra angular del procés de desenvolupament de programari, desplaçant així al codi font com a artefacte principal. Encara que aquest enfocament centrat en models ofereix avantatges sobre el desenvolupament de programari tradicional (per exemple, la generació de codi de forma automàtica a partir dels models) no té el nivell d'adopció esperat.

La literatura científica revela un ampli acord en el fet que els usuaris finals puguen ells mateixos desenvolupar i adaptar els sistemes però la complexitat dels estàndards de modelatge i la falta d'habilitats de modelatge impedeix la seua participació activa en processos DPDM existents. Per a aconseguir-ho, els usuaris finals han de disposar de llenguatges de modelatge diferents amb conceptes adaptats a les seues habilitats, context i necessitats.

Aquest desafiament és l'objectiu principal d'aquesta tesi que s'aborda combinant les idees del desenvolupament orientat a l'usuari final i el DPDM. Aquest treball comença involucrant usuaris finals en tasques de modelatge amb una eina que els proporciona un llenguatge de modelatge visual que permet als usuaris finals seleccionar i personalitzar característiques d'un sistema pervasiu utilitzant conceptes familiars per a ells. Després, aquesta tesi motiva la necessitat d'enriquir processos de DPDM existents per a suportar el desenvolupament d'una nova generació de sistemes programari (per exemple, salut intel·ligent) que requereixen coneixements especialitzats en una varietat de dominis. Conseqüentment, diferents tipus d'usuaris (per

exemple, científics, enginyers i usuaris finals) han de participar activament en la descripció de fragments de models que depenen de la seua experiència utilitzant un llenguatge de modelatge diferent. D'aquesta manera, els usuaris poden col·laborar per a obtenir una descripció del sistema unificada. En aquest punt, és necessari proporcionar mecanismes que transformen i integren els fragments d'un llenguatge de modelatge a un altre i delimiten quins fragments es descriuen per un usuari diferent.

Per a proporcionar açò, la proposta presentada utilitza la gestió de variabilitat de forma nova per a permetre modelatge col·laboratiu seleccionant fragments d'un model del sistema que poden ser descrits utilitzant un llenguatge de modelatge diferent i, la integració d'aqueixos fragments una vegada que hagen sigut descrits. A més, la proposta utilitza mecanismes d'interoperabilitat per a connectar dos llenguatges de modelatge diferents transformant la descripció dels fragments d'una manera no invasiva amb la seua estructura. Per tant, la nostra proposta pot enriquir els models de processos DPDM existents amb fragments de models que han sigut descrits amb un llenguatge diferent i açò, podria fer que els usuaris se senten segurs en adoptar models per a descriure contingut de domini específic i podria ajudar a adoptar processos DPDM.

La proposta ha sigut validada en tres casos d'estudi amb diferents nivells de complexitat i dominis: sistemes per a la llar intel·ligent, sistemes d'informació web i protocols biomecànics. Els resultats han demostrat l'aplicabilitat i viabilitat de la nostra proposta per a involucrar diferents tipus d'usuaris (usuaris finals amb professionals de programari, experts en el domini amb experts en desenvolupament de programari i, metges amb enginyers biomèdics, respectivament) en descripcions de models de processos DPDM existents utilitzant un llenguatge de modelatge diferent.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

This thesis addresses the involvement of users in modeling tasks by bringing the fields of End-user Development (EUD) and Model Driven Development (MDD) with the purpose of sharing knowledge in MDD processes from heterogeneous modeling languages.

Despite MDD is an established approach for developing software systems using models as the main ingredient of the development process, it has not been widely adopted [1, 2]. Nowadays, access to sophisticated models is restricted to a select few since users, who participate in software projects, face barriers and challenges (e.g., steep learning curves, arduous concepts and user interfaces) that make the adoption of models hard for them [1].

Furthermore, sharing knowledge in MDD processes continues to be a major challenge [3] even though it becomes necessary to develop a new generation of software systems, for example, smart health and intelligent transportation systems, that requires expertise in a variety of domains. Consequently, different types of users (e.g., scientists, engineers and end-users) must actively participate and collaborate describing different system aspects to obtain a unified software artifact [4] even they have different skills,

they often use different approaches and tools, and MDD processes have little effective support of collaborative modeling mechanisms to determine which concerns of the system rely on a different user [1].

Hence, the involvement of users in modeling tasks is not a closed research topic even though it is essential [5]. Users are often interviewed or in other ways heard [6] but they often lack the skills to transform their domain knowledge in models, which prevents that users are able to participate themselves in modeling tasks. Therefore, it is necessary to find ways to allow users to customize models themselves according to their particular skills, context and needs [1]. Thus, users could feel confident to adopt models, which could help to use MDD processes by the software industry as expected [1].

In this work, EUD brings the identification of both user skills and guidelines. The guidelines recommend the use of modeling languages that are closer to users' skills and specific modeling tools for actively involving users in modeling tasks [7, 8]. A closer modeling language may use concepts in which users are familiar and comfortable with in order to lower their barriers in the description of domain-specific content [7].

In order to address the active involvement of users in modeling tasks, we start with the involvement of end-users in an existing MDD process for developing pervasive systems by providing a tool-supported visual modeling language, which applies the EUD guidelines that are identified in this work. Thus, end-users are involved in modeling tasks by selecting and customizing system features. Next, we detected that the selection and customization of system features could limit the expressiveness of users, who may need to work on sophisticated and completed system descriptions. Then, we support the involvement of users by using a closer modeling language and integrating their descriptions into a modeling project of a MDD process. Therefore, different users actively participate in modeling tasks and share their knowledge to

obtain a modeling project that unifies all descriptions.

To reach this, we propose a method that combines non-intrusive interoperability, and variability mechanisms in a novel way to enable the description of model fragments using a different modeling language. By non-intrusive, we mean that the method does not modify the structure of modeling languages (meta-models). On the one hand, interoperability mechanisms solve the connection of model descriptions that are performed from different modeling languages by means of model transformations. On the other hand, variability mechanisms enable collaborative modeling scoping the user-dependent participation by allowing both the selection of model fragments as gaps of the system that may be described using a different modeling language, and the integration of those model fragments once they are described to obtain a unified system description.

The rest of this chapter is organized as follows: Section 1.1 motivates this thesis work. Section 1.2 states the problem that this thesis resolves. Section 1.3 defines the goals for this work. Section 1.4 describes the approach proposed in this thesis to fulfill the detected goals. Section 1.5 introduces the research methodology that has been followed. Section 1.6 explains the context in which the work of this thesis has been performed. Finally, Section 1.7 gives an overview of the structure of this thesis.

1.1 Motivation

MDD [9] is a software development paradigm that proposes using machine-readable models at various levels of abstraction as its main artifacts. The key idea is to automatically transform highly abstract models into more concrete ones from which an implementation can be generated in a straightforward way. Models are used to build software, thereby displacing source code

as the development process's main feature. The main benefits of models are that they enable reuse at the domain level, reduce costs by using an automated process, and increase the longevity of software solutions [9]. Models are expressed with a general purpose language or with a Domain-Specific Language (DSL) [10] and they describe concerns that refer from architecture to behavior, and also refer non-functional concerns such as security.

In this model-centric schema, modeler experts (from now onward modelers) usually carry out the description of concerns in models because they have the knowledge to express them by using specific technology such as modeling tools. Nevertheless, the description of concerns may also require the involvement of users from the very beginning [11, 12] since they know the domain and their needs to address a concrete problem.

For example, in the building domain, architects carry out the design of maps for a new customized house. Architects have the knowledge to describe engineering concerns in maps such as the design of the load girders but architects need the collaboration of a user to design a house that fulfills user's needs (e.g., a study room with a big window). Similarly, this example can be transferred to the description of concerns in models of a software system. Users may not be familiar with software engineering concerns, DSLs, and modeling tools but users have the knowledge about the problem domain [13].

The literature [6] reveals a broad agreement in the fact that it is important and useful to involve users in the construction and modification of models. In particular, end-users are generally neither skilled nor interested in adapting their systems at the same level as modelers but it is very desirable to empower end-users to adapt systems at a level of complexity that is appropriate to their individual skills and situations. This is the main goal of End-user Development (EUD) [14]: empowering end-users to develop and adapt

systems themselves.

Over the past quarter-century, considerable effort has been directed by researchers to involve end-users in the development of software systems. Currently, most end-users have become familiar with the basic functionality and interfaces of computers. However, developing new applications that effectively support end-users' goals still requires considerable expertise in programming that cannot be expected from most people [14]. Thus, one fundamental challenge for the coming years is to develop environments that allow end-users to develop or modify their own systems [14].

In the particular case of modeling tasks of MDD processes, the involvement of users is not a closed research topic even though it is essential [5] to the system success [15], industrial settings reveal a broad agreement in the fact that it is important and useful [16], and it could help to achieve a wider adoption of MDD processes in the industry [1, 4]. As introduced above, it is very important involving end-users in system conceptual descriptions because end-users have the knowledge about the problem domain [13]. In addition, end-users are more likely to successfully adopt and use the result if they are involved as partners in the design [17]. Nevertheless, involving users is a difficult task because they often lack modeling skills to deal with the complexity of current standard modeling concepts and tools to transform their domain knowledge in models as modelers do.

To overcome this complexity, users should be involved in modeling tasks using modeling concepts and tools that fit their particular skills, context and needs, but this emerges the necessity of exchange information among models of heterogeneous modeling languages in order to integrate users' descriptions into a common modeling project. Hence, model interoperability mechanisms are needed to involve users in modeling tasks of MDD process using a different modeling language that is closer to users' skills, and specific tool support is

also needed to allow users to participate themselves in model descriptions. These mechanisms may be domain-independent, and non-intrusive with the structure of models.

Model interoperability mechanisms is a growing trend [18] and it can provide several benefits, i.e., existing modeling languages can be used as complementary alternatives to perform the modeling tasks from different domains (such as system design, business processes, etc.), from different users (such as project managers, system designers, domain experts, etc.), and from different software representations that could have a different abstraction level (such as a visual language that helps the description of models). However, most modeling languages are not designed to interoperate with other modeling languages, which makes them isolated alternatives.

There are approaches such as [19, 20, 21] that achieve interoperability to perform modeling tasks from two different modeling languages. However, these approaches are focused on transformations from an entire model to another model, and these approaches do not provide collaborative modeling mechanisms to determine in which model concerns users may be involved.

1.2 Problem Statement

The interoperability between models of different modeling approaches for actively involving users in modeling tasks and integrating their descriptions in a modeling project, which unifies all descriptions in a non-intrusive way, is not a closed research topic. The above motivation indicates that some problems still need to be considered. The work presented in this thesis attempts to solve these problems, which can be stated by the following three research questions:

Research question 1. What user skills and software development activi-

ties are identified and which ones are going to be supported in modeling tasks?

Research question 2. How should users be involved in modeling tasks in an understandable way for them and what issues may be faced?

Research question 3. How should models interoperate to support users' descriptions from a different modeling approach in a non-intrusive and collaborative way?

These research questions are analyzed and answered in the following section.

1.3 Thesis Goals

The main goal of this thesis is to involve users in modeling tasks by enabling them to describe model fragments with a different modeling language and integrating users' descriptions into a modeling project of a MDD process in a non-intrusive way with the structure of such models. Thus, users can collaborate themselves, and they are guided in the creation or modification of model descriptions in order to obtain a unified modeling project, which is enriched with interoperability and collaborative modeling mechanisms.

First of all, regarding **research question 1**, one of the goals of this work is to involve end-users in modeling tasks of an existing MDD process. To achieve this, it is necessary to review the EUD field to identify the different skills that end-users can have on the basis of the development activities in order to establish the ones that this thesis supports. Furthermore, it is necessary to review the EUD field to identify a set of guidelines that make the participation of end-users easier in the description of system behavior in order to apply them for lowering barriers of end-users in modeling tasks of an existing MDD

process for developing pervasive systems. Thus, end-users can be actively involved in modeling tasks of this process in order to get a system description that fits their needs.

Regarding **research question 2**, another goal of this work to involve different types of users (e.g., scientists, engineers and end-users) in modeling tasks of a MDD process using a different modeling language. Thus, users are able to work on sophisticated system descriptions and avoid dealing with the description of engineering concerns and with complex concepts and modeling tools. To achieve this, it is necessary to identify both the phase of the MDD process in which different types of users may be actively involved and the issues that may be faced in order to propose a method that tackles them.

Regarding **research question 3**, one of the goals of the present work is to provide mechanisms that: (1) scope the user-dependent participation by delimiting which model fragments of the system should be described using models of a different modeling language, (2) translate model fragments from one modeling language to another for enabling interoperability between different modeling approaches, and (3) integrate those model descriptions into a unified modeling project in order to enrich models with descriptions of different types of users. These mechanisms may be non-intrusive (i.e., without affecting the structure of modeling languages), tool-supported, and domain-independent to favor their application in existing modeling languages of different domains.

1.4 The Proposed Solution

The solution that is proposed in this thesis enables that different types of users collaborate in modeling tasks by describing themselves model fragments using a different modeling language in a non-intrusive and collaborative way.

Specifically, the solution provides the following contributions:

1. The **identification of different user skills and guidelines** that the literature follows in order to involve end-users in software development activities, and their application for lowering barriers of end-users in the description of domain-specific content in models of an existing MDD process for developing pervasive systems. Since this existing MDD process requires skills to be involved in modeling tasks that most end-users lack, the application of the identified guidelines provides a tool-supported visual modeling language, which allows end-users to describe system properties that depend on them in models by selecting and customizing system features.

2. A method, **Medem**, which empowers different types of users with more expressiveness by supporting their involvement using a different modeling language, and integrates users' descriptions into the models of a MDD process by combining collaborative and modeling mechanisms. On the one hand, collaborative modeling mechanisms enable the selection of model fragments in a common modeling project to identify them as gaps. The description of these gaps is carried out by a user using a different modeling language. To make this feasible, variability mechanisms are used in a novel way to manage the creation and description of such gaps, which delimit the users' participation. On the other hand, users are provided with a customized view of the model fragments that they have to complete by themselves using a different modeling language. In this context, the use of model transformations is the cornerstone [22, 23, 24] of solving the connection among models of different modeling approaches. Therefore, users describe the gaps using a language closer to their knowledge, do not deal with complex concepts

and complex modeling tools, and are guided in modeling tasks.

3. **A model-based and variability-based implementation** that supports Medem using a variability model to manage gaps; and model queries, a weaving model and model-to-model transformations to achieve non-intrusive interoperability. This is supported in a transparent way to the users once a modeler initializes it.

The advantages of the proposed solution are the following:

1. Users are able to participate in modeling tasks using a modeling language according to their particular skills. This could make them confident to adopt models and lower barriers in the description of domain-specific content [7], which could promote the wider adoption of MDD processes.

2. Different modeling languages are able to interoperate in order to obtain the full description of a software system.

3. Users can be focused on describing concerns of the software system that depend on their knowledge rather than describe the entire software system.

4. The structure of the modeling languages is not modified. Thus, the proposed solution can be applied to existing modeling languages.

Finally, we conclude with the **empirical validation** of the proposed solution by applying three case of studies in order to involve different types of users (end-users and software professionals; domain experts and software development experts; and doctors and biomedical engineers) in different domains (smart home systems, web information systems, and biomechanical protocols, respectively), which have different levels of complexity. As a result, Medem has proven its applicability and feasibility.

1.5 Research methodology

In order to perform the work of this thesis, we will apply a research project following the design methodology for performing research in information systems as described by [25] and [26]. Design research involves the analysis of the use and performance of designed artifacts to understand, explain and, very frequently, to improve on the behavior of aspects of Information Systems [26].

The design cycle consists of 5 process steps: (1) awareness of the problem, (2) suggestion, (3) development, (4) evaluation, and (5) conclusion. The design cycle is an iterative process; knowledge produced in the process by constructing and evaluating new artifacts is used as input for a better awareness of the problem.

Following the cycle defined in the design research methodology, we started with the awareness of the problem (see Figure 1.1): we identified the problem to be resolved and we stated it clearly.

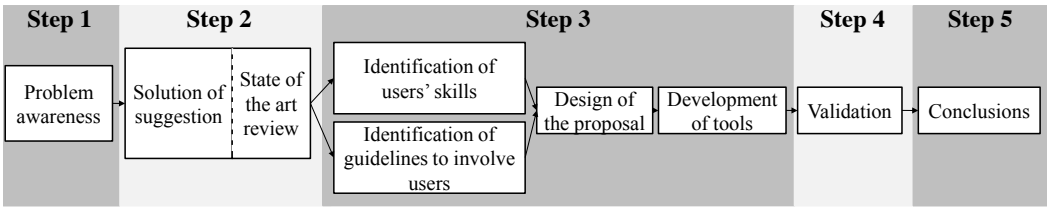


Figure 1.1: Research methodology followed in this thesis.

Next, we performed the second step that is comprised of the suggestion of a solution to the problem, and we compared the improvements that this solution introduces with already existing solutions. To do this, the most relevant approaches were studied in detail. Once the solution to the problem was described, we plan to develop and validate it (steps 3 and 4). These two steps will be performed in several phases (see Figure 1.1).

Finally, we analyzed the results of our work in order to obtain several conclusions and to provide some insights about further research (step 5).

1.6 Thesis Context

This thesis has been developing in the context of the research center *Centro de Investigación en Métodos de Producción de Software* of the *Universitat Politècnica de València*. The work that has made the development of this thesis possible is in the context of the following research government projects:

- SESAMO: Construcción de Servicios Software a partir de Modelos. CYCIT project referenced as TIN2007-62894.
- EVERYWARE: Construcción de Software Adaptativo para la Integración de Personas, Servicios y Cosas usando Modelos en Tiempo de Ejecución. CYCIT project referenced as TIN2010-18011.

Moreover, this thesis has been developing in the context of the *Universidad San Jorge* in conjunction with a full-time Lecturer position since 2012.

1.7 Thesis Structure

This thesis is comprised of ten chapters in total, which are organized as follows:

Chapter 2: *Background.* This chapter introduces the main fields to provide the reader a basic background for understanding the overall thesis work. In particular, this chapter presents End-User Development, Model Driven Development and Variability Management.

Chapter 3: *State of the Art.* This chapter presents the most well-known approaches for involving users in software development activities and

achieving MDD interoperability. To conclude, these approaches are compared with this work.

Chapter 4: *Addressing the Involvement of Users.* This chapter identifies the phase of the MDD process in which different types of users may be actively involved and the issues that may be faced in that phase. This chapter also shows the current practice in collaborative modeling, and overviews our envisioned approach through variability management.

Chapter 5: *Involving End-users in Modeling Tasks.* This chapter identifies different user skills and guidelines that the literature suggests in order to propose a solution that involves end-users in modeling tasks of an existing MDD process for developing pervasive systems by means of customizing system features using a tool-supported visual modeling language.

Chapter 6: *Achieving the Involvement of Users in Modeling Tasks with Heterogeneous Modeling Languages.* This chapter argues the necessity of involving different types of users in modeling tasks and providing them with more expressiveness in order to obtain a unified system description. In addition, this chapter presents our proposed solution for supporting collaborative modeling from a different modeling language.

Chapter 7: *Medem Tool Support.* This chapter presents the technological decisions that support the main building blocks of our proposed solution (interoperability and variability mechanisms).

Chapter 8: *Evaluation of the Proposal.* This chapter presents three case studies in different domains and levels of complexity (smart home systems, web information systems, and biomechanical protocols) and the application of the proposed solution for each one in order to involve different types of users in modeling tasks.

Chapter 9: *Towards the Efficient Specification of the Interoperability Mechanisms.* This chapter shows an extension of the proposed solution

that constitutes our ongoing work to achieve an efficient specification of the interoperability mechanisms. This chapter also shows the technological decisions to support this extension, and the application of the extension in the same three case studies to compare and discuss the results.

Chapter 10: *Conclusions and Future Work.* This chapter concludes by summarizing the main contributions, results and publications of this thesis work. In addition, this chapter provides some insights about further work.

Chapter 2

BACKGROUND

This thesis work relies on the fields of End-user Development, Model Driven Development, and Variability Management. In order to clarify the foundations of these fields, different concepts and techniques are introduced in this chapter.

First, the main concepts of End-user Development are presented. End-user Development aims to allow users, who are non-professional software developers, to develop or modify their own applications. In addition, representative techniques and metaphors of the End-user Development literature are overviewed to show how users can be involved in the description of system behavior.

Second, the main concepts of Model Driven Development are presented. Model Driven Development is a paradigm where models are becoming the new programming code to specify software products. The specification can be carried out using a general-purpose language or a DSL, which is specifically designed to a certain domain or a specific concern of a software system. Here, meta-modeling play an important role since the structure of models (concepts, relationships, and constraints) is defined in a model, which is

called meta-model. Moreover, meta-models are needed for the creation of model transformations, code generation, and tool integration.

Finally, the main concepts of Variability Management are presented since variability management aims to efficiently manage a range of products by specifying variable elements on them.

The remainder of this chapter is structured as follows: Section 2.1 introduces the main concepts and techniques of End-user Development. Section 2.2 introduces the main concepts of Model Driven Development, Domain-Specific Languages, meta-modeling, and interoperability of models. Section 2.3 presents Variability Management, its activities and facets. Section 2.4 concludes the chapter.

2.1 End-User Development

The fundamental aim of End-User development(EUD) [27] is to empower users to gain more control over their computers by involving them in a development process. To make this involvement useful, users often have to adapt these applications to their specific needs. Adaptation may assume many forms ranging from simple forms such as changing preference settings of applications, to more complex forms such as writing filtering rules for email applications or defining formulas for spreadsheets.

It is very important for users to actively participate in the development process because they have the knowledge about the problem domain [13]. Only the users of an application, not the developers of that application, can decide on how to deal with all the information that depends on the domain or their preferences. Therefore, application developers can no longer anticipate all the needs of users [28]. This discrepancy between what application developers can build and what individual users really need can be addressed

with EUD.

2.1.1 Definition

The term EUD is relatively new, but it stems from the field of End-User Programming (EUP) [28]. The shift from "programming" to "development" reflects the emerging awareness that, while the process of adapting a computer to the needs of a user may include some form of programming, it certainly is not limited to it. In that sense, most of the research questions from EUP carry over to EUD because of the widened scope of EUD new issues need to be explored. The EUD relevance is to potentially cover large segments of the population including most users of traditional computer applications but also of information technology associated with ubiquitous computing.

Users are generally neither skilled nor interested in adapting their systems at the same level as software professionals. However, it is very desirable to empower users to adapt systems at a level of complexity that is appropriate to their individual skills and situations. This is **the main goal of EUD: empowering users to develop and adapt systems themselves**.

Then, the most common definition for EUD is the following [14]:

> **End-User Development is a set of activities or techniques that allow people, who are non-professional software developers, at some point to create or modify a software artifact.**

Some existing research addresses this issue [14] , casting users as software professionals with the systems they are using. This clash between the two cultures becomes particularly evident when the system requires users to perform development activities [13].

EUD could lead to a considerable competitive advantage (economic)

of involve users as particular domain experts since users may be able of dynamically changing environments [29]. In addition, the increasing amount of software embedded within consumer and professional products also points a need of enabling EUD.

EUD is important for allowing full participation of citizens in the emerging Information Society. The Information Society enables access through a variety of interaction devices ranging from small mobile phones to large flat screens. However, the creation of content and the modification of the functionality are difficult for non-professional programmers, resulting for many sectors of society in a division of labor between those who produce and those who consume. EUD has the potential to counterbalance these effects.

Lieberman et al. [14] think that over the next few years, the goal of Human-Computer Interaction (HCI) will evolve from just making systems easy to use to making systems that are easy to develop. By now, most people have become familiar with the basic functionality and interfaces of computers. However, developing new or modifying applications that effectively support users' goals still requires considerable expertise in programming that cannot be expected from most people. Therefore, it is a challenge for the coming years the development of environments that allow users to develop or modify their own applications.

The emerging research field of EUD integrates different threads of discussion from Human Computer Interaction (HCI), Software Engineering (SE), computer supported cooperative work (CSCW), and artificial intelligence (AI). Concepts such as tailorability, configurability, end-user programming, usability, visual programming, natural programming, and programming by example already form a fruitful base, but they need to be better integrated, and the synergy between them more fully exploited.

2.1.2 Initiatives

The potential for designing a system that performs the wrong action and seriously annoys users is quite high [30]. Thus, the main goal for a system is to place it in the hands of users, so they can build and configure the system to do what they want when they want it. EUD well-accepted initiatives seek to achieve this main goal. Next, three techniques and three metaphors are described as follows:

Techniques:

Natural Programming. It is an application of the standard user-centered design process to the specific domain of programming languages and environments [31].

Myers et al. [31] claims that the premise of this approach is that programmers will have an easier job if their programming tasks are made more natural. By "natural", they mean "faithfully representing nature or life", which here implies it works in the way people expect. By "natural programming" they are aiming for the language and environment to work the way that non-programmers expect. Thus, the Natural Programming goal is to make possible for people to express their ideas in the same way they think about them.

The Natural Programming design process, that treats usability as a first-class objective, follows these steps:

1. Identify the target audience and the domain, that is, the group of people who will be using the system and the kinds of problems they will be working on.
2. Understand the target audience, by studying the actual language, techniques, and thinking they naturally use when trying to solve

problems. This includes an awareness of general HCI principles as well as prior work in the psychology of programming and empirical studies. When issues or questions arise that are not answered by the prior work, conduct new user studies to examine them.

3. Design the new system based on this information.
4. Evaluate the system to measure its success, and to understand any new problems the users have. Redesign the system based on this evaluation, and then reevaluate it, following the standard HCI principle of iterative design.

Natural Programming has significant importance for EUD since it provides a methodology model that can be followed by other developers and researchers when designing their own languages and environments. Myers et al. [31] believe this will result in more usable and effective tools that allow both end-users and professionals to write more useful and correct programs.

Programming By Example. It is also called Programming by demonstration because the user demonstrates examples of the desired behavior to the computer [32]. Programming By Example (PBE) was introduced by Smith in the mid-seventies [33] and it consists of demonstrating desired computational behavior via concrete examples by the end-users, rather than in the form of abstractions (eg., programming code) [34]. Originally Programming By Example was aimed solely at single desktop environments but recently a number of researchers have started to explore how these ideas might be applied to digital homes, made up of distributed embedded computers (usually integrated into household appliances).

Currently most end-user programming tools for pervasive applications

are based on a procedural programming metaphor and require that the user mentally manipulate constructions, which would be familiar to most programmers (e.g., albeit in a graphical or macro form) thereby placing a significant cognitive load on the user.

Programming By Example systems have two levels of representation [35]:

1. The GUI level. This level features familiar windows, icons, and menus. For instance, in a word processing application, such as Word, this level represents content directly manipulated by users with such operations as the typing of new text, the formatting of text, and the use of cursor keys to navigate through a document.
2. The program level. This level captures user manipulations directly into programs so users can replay them. In Word, this level incorporates Visual Basic. User manipulations are recorded as Visual Basic scripts; users then assign scripts to keyboard commands or to user-defined toolbar commands.

Programming By Example can be used as a mixed-initiative for active learning. The mixed-initiative approach uses agents and graphical widgets to obtain input from a user in order to both help a recognizer improve its recognition ability and to resolve ambiguity. Similarly, active learning systems make queries to the user or perform experiments to gather data that are expected to maximize performance. Active learners demonstrated significant decreases in the amount of data required to achieve the equivalent performance of passive learners [30].

As mentioned before, Programming By Example is not the only approach available to involve end-users but Dey et al. [30] believe that Programming By Example offers long-term potential for supporting

dynamic and complex behaviors. This is because Programming By Example allows end-users to build context-aware behaviors in a situated manner that would otherwise be too complex or time consuming to build.

Although Programming By Example is an interesting technique to empower end-user to customize their systems, it is not a widespread technique. This is because it represents a radical departure from what it is known as programming. The conservatism of the programming community is the biggest obstacle to widespread Programming By Example use [32].

Visual Programming. Visual Programming (VP) refers to any system that allows the user to specify a program in a two (or more) dimensional fashion [36]. VP uses information in a format that is closer to he user's mental representations of problems, which allow data to be processed in a format closer to the way objects are manipulated in the real world. It seems clear that a more visual style of programming could be easier to understand and generate for humans, especially for non-programmers or novice programmers [37]. Moreover, the use of graphics tends to be a higher-level description of the desired actions (often emphasizing issues of syntax and providing a higher level of abstraction) and therefore, it may make the programming task easier even for professional programmers. Also, some types of complex programs, such as those that use concurrent processes or deal with real-time systems, are difficult to describe with textual languages, so graphical specifications may be more appropriate.

Metaphors.

Jigsaw Metaphor [12]. It is based on the familiarity evoked by the notion

and the intuitive suggestion of assembly by connecting pieces together. Essentially, the jigsaw metaphor allows users to connect components and compose various arrangements through a series of left-to-right couplings of pieces. Constraining connections in a left to right fashion also provides users with the sense of a pipeline of information flow. The Jigsaw metaphor is a rule-based metaphor.

Magnetic Poetry [38]. It consists of small, flexible individual magnets, each of which has a word printed on it. Users can combine the words into "poem" or statements to a variety of effects. Magnetic poetry sets often have a theme or topic, such as "love" or "computers" and contain words related to that theme, the resulting poems are geared towards that topic.

Butler [11]. It promotes the interaction of the user and the system by means of instructing a butler. The input modality is a spoken dialogue with the system. For example, if the user wants turn on the kitchen light, the user had to tell a butler: "switch on the kitchen light" and then, the system switch on the kitchen light. Previously, the user have to program the butler actions and, when the system is in run-time, the user should remember some basic information to use the system like a butler. The Butler metaphor is a rule-based system.

It is important to highlight that most EUD tools, which involve users in programming tasks, are currently based on metaphors [34], which require users to mentally manipulate constructs that they are familiar with.

2.2 Model Driven Development

Model Driven Development (MDD) [9] is a recent paradigm that leds a dramatic change in the software development process since models have become the main artifact in the development [39]. Consequently, models are becoming the new programming code in a process that is driven by model specifications and by transformations among models. Thus, it is obtained the artifact implementation.

Model Driven Architecture (MDA) is a framework for software development proposed by the Object Management Group (OMG) in 2001[40] (i.e., MDA is a concrete realization of MDD). The OMG is a consortium of software vendors and users form industry, government, and academia. The notion of Model Driven Engineering (MDE) emerged later as a paradigm generalizing the MDA approach for software development [41].

2.2.1 Definition

MDD and MDA are changing the development process of software since their main distinguishing feature is to establish models as products rather than programming code. Models are defined as follows [40]:

> **A model of a system is a description or specification of that system and its environment for some certain purpose. A model is often presented as a combination of drawings and text. The text may be in a modeling language or in a natural language.**

Models are present throughout the software development course of understanding, design, construction, deployment, operation, maintenance and modification. As stated by Agrawal [42]:

> "the models are not merely artifacts of documentation, but living documents that are transformed into implementations. This view radically extends the current prevailing practice of using UML: UML is used for capturing some of the relevant aspects of the software, and some of the code (or its skeleton) is automatically generated, but the main bulk of the implementation is developed by hand. MDA, on the other hand, advocates the full application of models, in the entire life-cycle of the software product."

Therefore, the goal of these approaches is to automatically translate models (an abstract specification of the system) into a fully functional software product. The major advantage of this is that models could be both less sensitive to the chosen technology (platform-independent) and much closer to the problem domain with regard to the most popular programming languages [43].

2.2.2 Initiatives

Models have been used for a long time in the software development field. Ranging from formal and executable specification languages (like OBLOG [44], TROLL [45] or OASIS [46]), to the most accepted notations (like UML [47]) and processes (like RUP [48]) models are present in the software development area.

Stuart Kent [41] defines Model Driven Engineering (MDE) by extending MDA with the notion of software development process (that is, MDE emerged later as a generalization of the MDA for software development). MDE refers to the systematic use of models as primary engineering artifacts throughout the engineering lifecycle. Kurtev provides a discussion on existing MDE processes [49] (refer to [50, 51] for a specific approach). In general, these

approaches introduce concepts, methods and tools [52]. All of them are based on the concept of model, meta-model, and model transformation.

Model Driven Architecture (MDA) is a concrete realization of MDD. MDA classifies models into two classes: Platform Independent Models (PIMs) and Platform Specific Models (PSMs) [40]. A PIM model is a view of a system that contains no specific information to the platform, or the technology that is used to realize it. On the contrary, a PSM is a view of a system that contains details of the particular type of platform that is used. Doing so, the definition of platform becomes fundamental, which is defined as [40]:

> "A platform in general is a set of subsystems/technologies that provide a coherent set of functionality through interfaces and specified usage patterns that any subsystem that depends on the platform can use without concern for the details of how the functionality provided by the platform is implemented."

Although the contribution of MDA has been critical, other initiatives under different descriptive terms have pushed on the Model-Driven Software Development (MDSD) direction. These initiatives (or specific paradigms) highlight distinct aspects and/or follow specific strategies for applying MDSD. The following are remarkable examples of these initiatives.

Automatic programming. According to Balzer [53], who is considered the initiator of the modern automatic programming paradigm, automatic programming is based on the use of methods and tools that support the acquisition of high level of abstraction specifications, their validation and the generation of executable code. He was focused on the generation of efficient implementations, since the hardware resources (CPU power, memory size, etc.) were limited. Therefore, he proposes a semi-automated (interactive) translation approach which facilitates the

specification of optimizations by human developers.

Generative Programming. This paradigm was proposed by Czarnecki in his PhD Thesis [54] although the term was coined by Eisenecker in [55]. In Eisenecker words, Generative Programming "is a comprehensive software development paradigm to achieving high intentionality, reusability, and adaptability without the need to compromise the runtime performance and computing resources of the produced software". It uses techniques like generic programming, and domain-specific languages.

In general, MDSD initiatives promote a paradigm of reuse and automation. This emerges through the extensive use of models and model transformations, which replaces cumbersome (and usually repetitive) implementation activities. In this way, model-driven approaches improve development practices by accelerating them.

2.2.3 Domain-Specific Languages

Domain-specific languages (DSLs) play a key role in several of the MDSD approaches that have been presented above. A DSL is defined as follows [56, 57]:

> **A DSL is a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain.**

DSLs are not a new topic, for example, a DSL for programming numerically-controlled machine tools, was developed in 1957-1958 [58]. On the last years, the current stress on MDSD has focused the interest of both

academy and industry on this kind of languages. Examples of DSLs abound, including well-known and widely-used languages such as LATEX, YACC, Make, SQL, and HTML. As state by [56], the older programming languages (Cobol, Fortran, Lisp) all came into existence as dedicated languages for solving problems in a certain area (business processing, numeric computation, and symbolic processing, respectively).

DSLs are tightly related to the Domain Engineering. In words of Tolvanen [59], the main focus of Domain Engineering is finding and extracting domain terminology, architecture and components. It is important to note that two points of view when dealing with the domain concept can be considered, as highlighted by Simos [60].

Conceptual domain. From this point of view, a domain is a set of interrelated real-world concepts. For instance, the health-care domain contains concepts like medical center, patient, disease, medicament, etc. As another example, the industrial factory domain contains concepts like stock, supplier, client, worker, etc.

Systems domain. From this point of view, a domain is characterized by a set of systems that share some common features [60]. These systems usually address a common problem area and conceivably share a common solution structure. In this case, we can talk about the expert systems domain, the database-based systems domain, the control/monitoring systems domain, the software games domain, etc.

Note that a software system can be seen as the combination of both a conceptual domain and a system domain. For instance, we can find experts system for health-care and control/monitoring systems for industrial factories, but also exists expert systems for industrial factories and control/monitoring

systems for health-care. Specific languages exist both for conceptual domains and systems domains.

Many benefits can be found in the literature about using DSLs. For instance, according to [56, 58].

- DSLs offer substantial gains in expressiveness since it is tailored to a specific problem domain.
- DSL programs are concise, and they can be reused for different purposes.
- DSLs enhance productivity, reliability, maintainability, and portability.
- DSLs embody domain knowledge, and thus enable the conservation and reuse of this knowledge.
- DSLs allow validation and optimization at the domain level.

Nevertheless, some drawbacks have been also identified about using DSLs. These drawbacks are related to the associated costs (for designing, implementing and learning the DSL) and the specific nature of the language (possible lack of expressiveness and/or loss of efficiency).

Some researchers and the EUD community claims the use of DSLs with visual notations seems to be the best option since visual languages have demonstrated to be more intuitive and easier to use than other options like textual languages [7]. This is known as Domain-Specific Visual Languages (DSVL), which are contingent on making similar tools and concepts for visual languages. Thus, the gap between the mental model of the user and the concepts of the DSVL are lower, and DSVLs can be understood by a wide audience. As a result, DSVLs can lower the initial hurdle to adoption [61].

2.2.4 Meta-modeling

Meta-modeling is one of the most important techniques of MDD [62] since it is used for producing meta-models. For example, meta-models are needed for dealing with construction of Domain-Specific Languages (DSLs). Moreover, meta-models are needed for the creation of model transformations, code generation, and tool integration. A meta-model is defined as follows [63]:

A meta-model is a model that defines the language for expressing a model.

The meta-model contains an abstract description of the structure of models by including the concepts, how these concepts are related, and constraints that have to be respected in the domain (abstract syntax). In order to create models, the concepts of the meta-model have to be represented (whether graphical or textual) using a concrete syntax in a model editor.

Figure 2.1 shows an overview of the meta-modeling architecture. Note that, as the right-side of the figure shows, this distinction corresponds to the usual distinction between concrete and abstract syntax. It is important the distinction between abstract and concrete syntax because just the meta-model (abstract syntax) is used as basis for dealing with interoperability among models, code generation, and model transformations.

The upper-left side of Figure 2.1 shows, the presentation of the models in a model editor interface (e.g. textual, tree-like, or diagrams) while the models are located at the logic of the bottom-left side of the figure. In addition, the logic shows the model processor to refer all the tasks related with handling models (e.g. code generation, or model transformations). This module is based on the ideas expressed in [64, 22] and represents any task whose input or output is a model. For instance, when a user requests a model to be transformed to another one. Also, the white arrow shows the connection

among the presentation, the model and its meta-model, whereas that the shadow arrow shows the connection between the model (and also the meta-model) with the model processor.

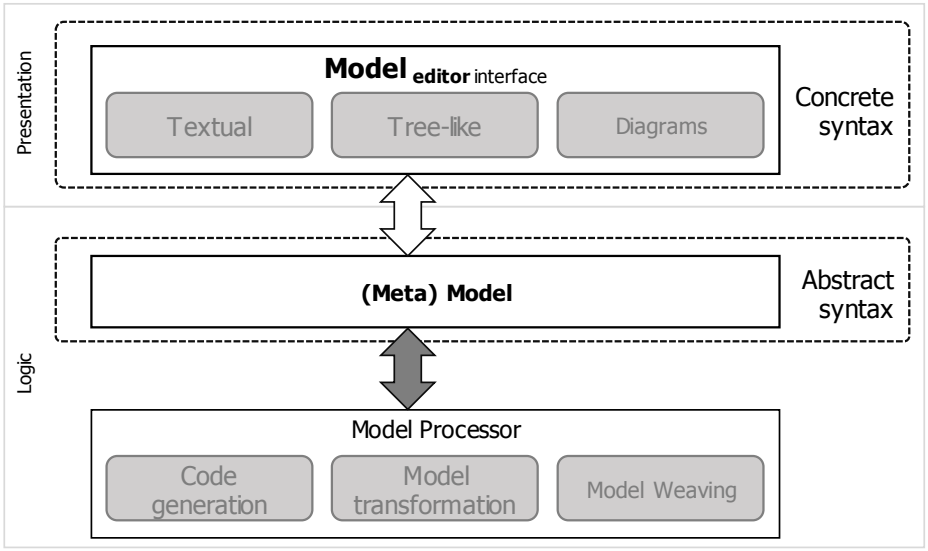


Figure 2.1: Overview of meta-modeling conceptual architecture

In order to define a meta-model, a meta-modeling language (defined by a meta-meta-model) and its tools are required. Meta-models have a class-instance relationship with models, which each model is an instance of a meta-model. This is also known as a model conforms to its meta-model, when the model uses the concepts that were defined in the meta-model, and it fulfills the meta-model relationships and cardinality constraints. This definition process belongs to the four-layered architecture designed by MOF [63].

Figure 2.2 shows the overall organization of an approach for defining a DSL and connections among the *Modeler expert* (from now onward modeler, who represents a role with modeling skills) and elements. Moreover, the upper side of the figure shows the distinction between the specification and execution phases. This distinction is based on the ideas expressed in [65] which divide the process into the specification of the DSL and its use by

creating models, respectively.

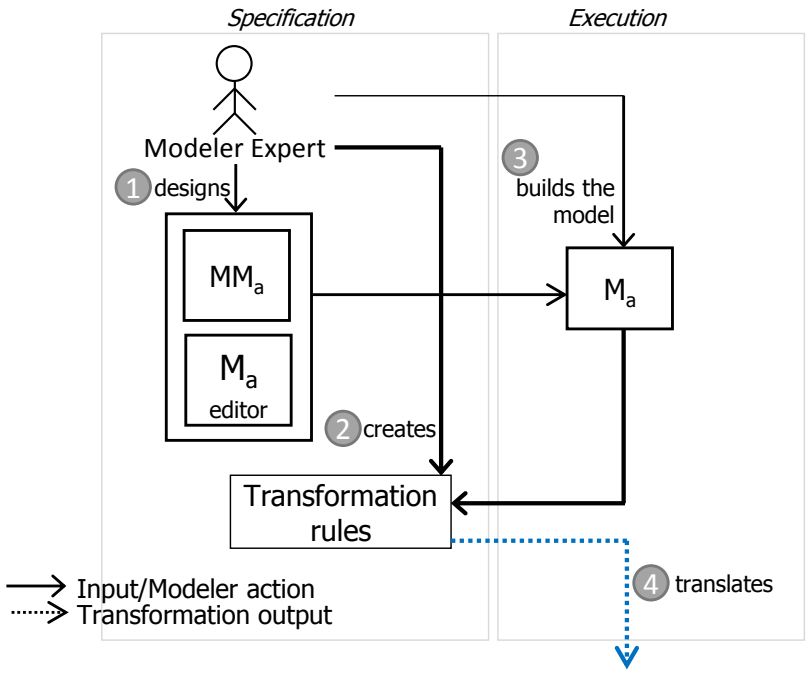


Figure 2.2: Overall MDD approach

For example, a simple meta-model for describing smart home systems can be specified by the *Modeler expert* using the meta-modeling tools, with a class given a name, *Service* and a number of attributes, such as *name* and *description*. The representation (graphical or textual) of the meta-model concepts (concrete syntax) is shown in an editor, which is the interface of the *Modeler expert*. The editor can be automatically generated by meta-modeling tools taking as input the specification of the abstract and concrete syntax (see Figure 2.2(1)). Then, the *Modeler expert* specifies the model processor tasks (i.e., transformation rules as Figure 2.2(2) shows). Next, the *Modeler expert* builds a model$_a$ that conforms the MM$_a$ using the M$_a$ editor (see Figure 2.2(3)). For example, the *Modeler expert* uses the editor to create a model (as an instance of the previous meta-model), usually in the execution, with a service with the name *KitchenLightsOn*, and the description: *this service*

switches on the light located in the kitchen. At the end, the editor stores the *Modeler expert*'s description in the model$_a$ and it can be processed. For example, transformation rules can be performed in order to translate the M$_a$ (see Figure 2.2(4)).

2.2.5 Interoperability of Models

In a MDD process, the system description can be collected in different models [22] in order to provide complementary alternatives to perform the modeling tasks. According to the IEEE Standard Computer Dictionary [66], interoperability is defined as follows:

> **Interoperability is the ability of two or more systems or components to exchange information and to use the information that has been exchanged.**

Interoperability of models can provide several benefits [18], i.e., existing modeling languages can be used as complementary alternatives to perform the modeling tasks from different domains (such as system design, business processes, etc.), from different roles (such as project managers, system designers, domain experts, etc.), and from different software representations that could have a different abstraction level (such as a visual language that helps the description of models). For this reason, it is necessary that different models interoperate in order to exchange system descriptions.

In order to support the interoperation of different models, the *Modeler Expert* has to develop a battery of model transformations [22, 51] to connect them. Thus, a source model that conforms to a meta-model can be translated into a target model that conforms to a different meta-model. The use of model transformations is a useful and efficient way of solving the connection of models to interoperate [22, 23, 24].

Many specialized solutions and approaches for model transformations exist [65], ranging from: textual [67, 68] to visual [69, 68, 70]; declarative [69, 70] to imperative through hybrid [67, 68]; and semi-formal [67, 68] to formal [69, 70]. This work is focused on a hybrid approach since it is the most followed by the most adopted languages [22] and it becomes popular and useful tools in research and industry [71].

Figure 2.3 shows the overall organization of a hybrid approach for model transformations. In particular, the figure shows elements and their connections for achieving interoperability of models that conform different meta-models and they are described from different users, who each one represents a user with different knowledge and skills. The left side shows the required input to the specification of model transformations: a meta-model and an editor for the source and target model. The center side shows the specification of mechanisms for supporting the transformation from a source model to a target model, whereas the right side shows the execution to build and modify models. Next, we explain the steps of this approach corresponding to the different numbers shown in the figure.

Step 1. First, the *Modeler Expert* designs the schema mapping. The schema mapping relates elements between two heterogeneous Meta-Models. For example, Meta-Model$_a$ that describes the structure of a DSL$_a$ (see MM$_a$ in Figure 2.3) and Meta-Model$_b$ that describes the structure of a DSL$_b$ (see MM$_a$ in Figure 2.3). The relationships are based on some semantic similarity of the concepts and are usually called correspondences. These correspondences are usually stored in a special kind of called the Weaving model. There are many previous works such as [72, 73, 74, 24] that design a Weaving model to bridge concepts between two heterogeneous Meta-Models in order to provide tool interoperability. The weaving model is designed before the system

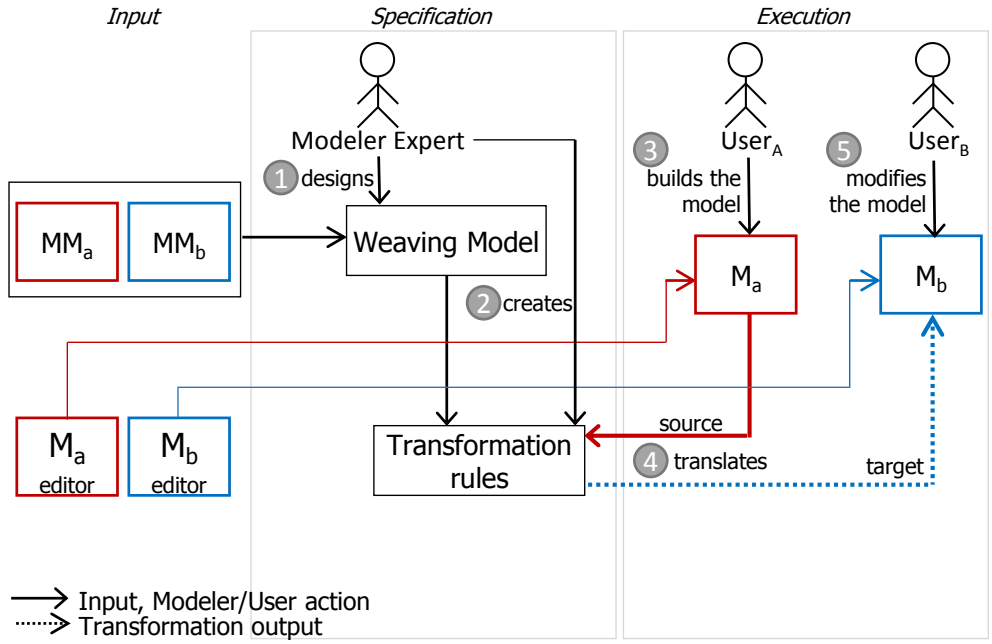


Figure 2.3: Overall hybrid approach for model transformations

description is carried out, so correspondences are established from each concept of the Meta-Model$_a$ to concepts (one or more) of the target Meta-Model$_b$. Note that only only those concepts that have any correspondence in the weaving model will be translated. Therefore, this step may require that the *Modeler Expert* spends some effort in stabling all the correspondences between meta-models, specially if the source meta-model has a high number of concepts.

Step 2. Once the weaving model is designed, the *Modeler Expert* takes as input the weaving model for creating transformation rules. Transformation rules can be extracted into a textual language (i.e., the Atlas Transformation Language [67]) to be executed in a specific transformation engine. Thus, the transformation rules are used to transform the models (see M$_a$ in Figure 2.3) conforming to the input meta-model (MM$_a$) into the model (see M$_b$ in the figure) conforming

to the output meta-model (MM_b). Although there are approaches such as [39, 75, 76] that seek the improvement of modelers' productivity by reducing the specification effort of model transformations by means of semi-automatic approaches and generic model transformations, model transformations are mostly created manually and adhoc [76].

Step 3. In the execution, the $User_a$ builds a $model_a$ (see M_a in Figure 2.3) conforming to the Meta-$Model_a$ using a $model_a$ editor. $User_a$ represents a user who has knowledge in Meta-$Model_a$ concepts and how the Meta-$Model_a$ concepts are represented (concrete syntax).

Step 4. Once the $model_a$ is built, the transformation engine automatically translates the entire source $model_a$ to a target $model_b$ conforming to the Meta-$Model_b$ by using the transformation rules that were previously created in Step 2.

Step 5. Finally, the $User_b$ may check and modify the $model_b$. $User_b$ represents a user who could have different skills with regard to $User_a$ since $User_b$ has knowledge in different concepts (Meta-$Model_b$ concepts) and the representation of those concepts in the $model_b$ editor.

Note that the specification of both the weaving model and transformation rules is a challenging task [39] that demands the most *Modeler Expert*'s time and effort. The specification of the weaving model (Step 1) is mostly manual [76] and its development time increases as the number of transformations grows due to the number and complexity of the meta-model concepts [39]. The specification of the transformation rules (Step 2) is often semi-automatic to reduce the development time [39] but it may also require manual refinements.

2.3 Variability Management

Over the decades, variability has become increasingly important in software engineering. Whereas software systems originally were relatively static, this is no longer acceptable for contemporary software systems [77].

Therefore, the necessity for modeling variability of software systems has been realized recently [78, 79] to efficiently manage a range of products by specifying their variable elements. For this reason, Software Product-Line Engineering (SPLE) [80, 81, 82] has gained significant attention over the recent years. It is claimed that SPLE provides a promising way to develop a large range of software-intensive systems faster, better, and cheaper [81]. Variability Management is a fundamental activity in SPLE [83]. It is also considered one of the key feature that distinguishes SPLE from other software development approaches or traditional software reuse approaches [84].

2.3.1 Definition

Variability refers to the ability of an artifact to be configured, customized, extended, or changed for use in a specific context [85]. Whereas variability management is defined as follows [86]:

> **Variability Management is the set of activities aimed to cover the creation and support of differences in versions of reference processes.**

Variability Management encompasses the activities of explicitly representing variability in software artifacts throughout the life cycle, managing dependencies among different variabilities, and supporting the instantiations of those variabilities [87]. Specifically, Variability Management activities are divided in two phases throughout the life cycle: domain engineering and application engineering. Domain engineering covers variability modeling

(identification, implementation and maintenance of variable elements in a model). Application engineering is responsible for making specific choices for the variable elements (resolution). An automated model transformation translates the resolution to the implementation. Thus, a specific product is obtained.

Two important concepts related to variability are variation points and variants. Variation points are locations in the design or implementation at which variation will occur, and variants are the alternatives that can be selected at those variation points [88].

There are two facets [79] in modeling variability. First, there are approaches that use feature models to describe variability of products. For example, Voelter and Groher [89] specify a feature model that describes the identification of variants and the combinations of features that produce valid variants in the domain engineering activity, and they select among a fixed number of predefined features to obtain a specific product in the application engineering activity. Second, the variability identified in products must have models that describe them. For example, Haugen et al. [90] propose the Common Variability Language (CVL) to specify gaps (placement fragments) as variants in a model in the domain engineering activity, and they fulfill those gaps in the application engineering activity by describing them in models (replacement fragments) that fit into gaps. The next subsections present these two facets in modeling variability and their concepts in detail.

2.3.2 Features for managing the variability of products

Since its first introduction in 1990 [91], feature modeling has been the most popular technique to model commonality and variability of products of a product line [92]. In fact, feature modeling has been become in the de facto standard for modeling software product lines. Commonalities and variabilities

are modeled from the perspective of product features. Thus, a feature model represents the information of all possible products of a software product line in terms of features and relationships among them as a hierarchically arranged set.

To start with, there are basic feature models [93], which their concepts support the creation of variants and relationships among variants. Figure 2.4 depicts a simplified basic feature model that shows how features are used to specify variants of watches. This example is inspired by an existing example [94]. The basic feature model concepts are described as follows:

- **Feature.** It is a product component that can be captured as common or variant. For instance, the *Display* feature.
- **Mandatory.** It is a relationship that includes a child feature in which feature parent appears. In the example, the *Display* feature is mandatory for watches.
- **Optional.** It is a relationship that a child feature can be optionally included in which its parent feature appears.
- **Alternative.** It is a relationship that only one child feature can be selected when its parent is part of the product. In the example, only one of the child features of *Display* can be selected.
- **Or.** It is a relationship that one or more child features can be selected when its parent is part of the product. In the basic feature model of Figure 2.4, whenever *Logical watch* is selected, *Alarm clock*, *WorldTime* or both can be selected.

In addition, a basic feature model can be extended to contain the following cross-tree constraints between features:

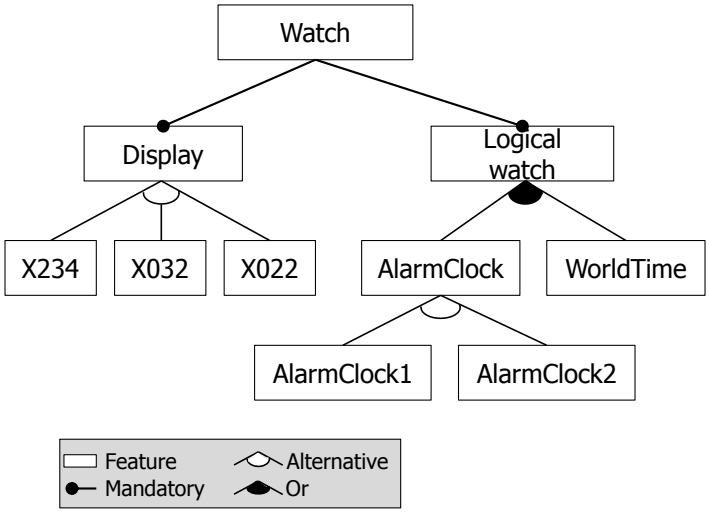


Figure 2.4: A feature model example to specify variants of watches

- **Requires.** It is a relationship that implies the inclusion of a feature B if a feature A that requires B is selected.

- **Excludes.** It is a relationship that implies the exclusion of a feature B if a feature A is selected. Therefore, both features cannot be selected in the same product.

Moreover, feature models can be extended with cardinalities to introduce new concepts as follows:

- **Feature cardinality.** It is a relationship with lower and upper bound denoted [n,m] that determines the number of instances of the feature that can be part of a product. This is a generalization of the above described *Mandatory* ([1,1]) and *Optional* ([0,1]) relationships.

- **Group cardinality.** It is a relationship with lower and upper bound denoted <n,m> that determines the number of child features that can be part of a product when its parent feature is selected. For example, a group cardinality that is equivalent to the *Alternative* relationship described above is <1,1>.

Besides, it is sometimes necessary to extend feature models to include more information about features such as cost required to support the feature. These models are *attributed* feature models. These attributes can be used to specify extra-functional information and they should consist at least of a name, domain and a value according to most proposals [93].

The main advantage of feature models is that they provide a clear overview of the variability and commonalities within a system. Nevertheless, feature models limit the expressiveness [95] in the application activity since feature models can only provide a bounded selection of features to obtain a product.

2.3.3 Models for managing the variability of products

In this facet of modeling variability using models to describe variation points rather than features, two approaches are distinguished [96, 90]:

- **Annotating the base model by means of extensions to the base modeling language.** The advantage of this approach is that it marks those model elements in which variation may occur, while the disadvantage is that base models are intimated with variability specifications.

- **Using a separate variability language**. Thus, variability models can be produced after the DSL is put into production without any modification to the DSL itself or the supporting tools. Since the variability is represented in separated models, more than one set of variation points and resolution rules can be expressed for each base model (this model is also known as variability model). For example, the project MoSiS there is a language CVL (Common Variability Language) [97] for modeling variability that follows the separate language approach.

CVL is a proposal sent by IBM, Thales, Fraunhofer FOKUS and TCS for the OMG Common Variability Language (CVL) [97] Request For Proposal (RFP), the CVL concepts are described in a nutshell.

Figure 2.5 shows an equivalent CVL model for the watch example above presented. The concepts of the variability model are the following:

- **Placement fragment**. Any model element or set of model elements that are variation points (gap). In the watch example, a Placement fragment for the *Alarm clock* is created in order to set it as variable.

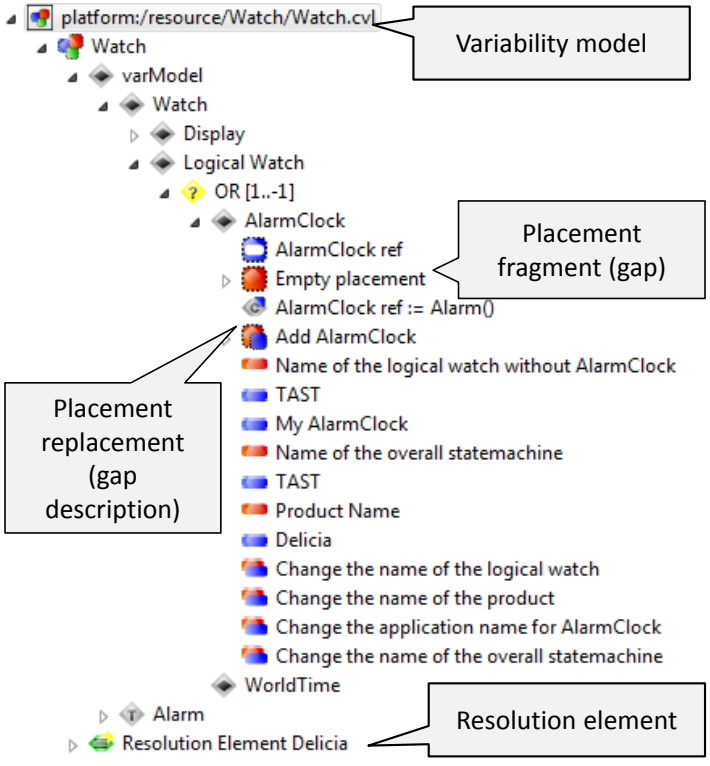


Figure 2.5: A CVL model example to specify variants of watches

- **Replacement fragment**. A model element or a set of model elements (gap description) that can be used as variant for a Placement fragment. In the example, an alternative *Alarm clock* (*AlarmClock1*) is created as

Replacement fragment. Moreover, more Replacement fragments can be created to set more *Alarm clock* variants (*AlarmClock2*).

- **Substitution**. A link between a placement and a replacement. This link also stores **boundaries**, which indicate other model elements that are inside or outside each fragment (placement or replacement). For instance, a Substitution is created to link the *Alarm clock* Placement fragment with the *AlarmClock2* Replacement fragment.

- **Resolution**. The specific replacement choices for placements. In the example, the watch is set to have the *AlarmClock2* alarm clock, so a Resolution element is created to store this choice for the *Alarm clock* Placement fragment.

The reader is referred to [94, 98] for a detailed description of CVL concepts.

2.4 Conclusions

The goal of this chapter was to provide a brief introduction to the existing background in which this work is built on. **End-user Development** addresses the discrepancy between what application developers can build and what users really need by means of their involvement in the creation or modification of software artifacts. **Model Driven Development** is a paradigm to develop programs based on models rather than programming code, which improve the development practices by accelerating them. **Variability Managament** is a set of activities for representing alternatives in a range of products, which provides a promising way to develop the software systems faster, better, and cheaper.

Acronyms You Need

EUD: End-User Development is a set of activities or techniques that promote the involvement of users, who are non-professional software developers, to create or modify a software artifact.

VP: Visual Programming uses information in a format that is closer to users' mental representations of problems. Thus, the information could be easier to understand and generate for users.

OMG: The Object Management Group is an international, not-for-profit industrial consortium that creates and maintains software interoperability specifications.

MDA: The Model-Driven Architecture is a set of OMG standards that enables the specification of models and their transformation into other models and complete systems.

MDD: Model Driven Development is an emerging paradigm for software construction that uses models to specify programs, and model transformations to synthesize executables.

MM: a Meta-Model contains an abstract description of the structure of models by including the concepts, how these concepts are related, and constraints that have to be respected.

XMI: The XML Metadata Interchange is an OMG standard for exchanging metadata information via Extensible Markup Language (XML). The most common use of XMI is as an interchange format for UML models, although it can also be used for serialization of models of other languages (meta-models).

DSL: A domain-specific language is a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain.

DSVL: A Domain Specific Visual Language is a DSL with visual notations, which have demonstrated to be more intuitive and easier to use than other options like textual languages.

SPLE: A Software Product Line Engineering is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way.

CVL: The Common Variability Language expresses variability in a language independently of the base modeling language. This base-model can be a DSL as well as a general purpose languages like UML.

Chapter 3

STATE OF THE ART

Some modeling approaches target application scopes which are intimately related to users' everyday activities. However, users cannot be involved in modeling tasks even though they are the ones who best know the expected functionality in a concrete domain. This is because users must have certain software development skills to capture every important aspect of their software system through models and primitives that *Modeler experts* usually use. For example, both PervML [99] and Habitation [100] are MDD processes that target the services of smart home systems using DSLs that can only be designed by *Modeler experts* due to the required knowledge of technologies (i.e., UML, Action Semantic Language (ASL), or Object Constraint Language(OCL)).

Giving users ways to easily customize behavior in well-specific domains, or customize their own tools to develop their daily work activities is important and it is a complicated matter, especially in modeling approaches in which models are used as main artifacts and their construction may require skills that some involved parties lack.

This chapter analyses approaches found in the literature by classifying

them in two categories that are related to the goals of this work. The first category includes approaches for involving users in the creation or modification of a software artifact in the pervasive systems domain and different ones since the first goal of this work is to provide mechanisms that involve end-users in modeling tasks. The second category includes approaches for achieving interoperability between modeling approaches since the second and third goals of this work seek to use different modeling approaches to involve different types of users.

The rest of this chapter is organized as follows: Section 3.1 presents the analysis criteria that classifies and analyzes the approaches found in the literature. Section 3.2 and Section 3.3 analyze the approaches for involving users and achieving MDD interoperability, respectively. Finally, Section 3.4 summarizes and discusses the analyzed approaches, and concludes the chapter.

3.1 Analysis Criteria

This section explains the classification criteria and features that we have considered relevant to manage and analyze approaches found in the literature. In order to manage and classify the approaches, we identify two categories as follows:

1. **Involving Users.** It includes approaches that follow EUD techniques and metaphors.

2. **Achieving MDD Interoperability.** It includes approaches that achieve interoperability of models.

In order to analyze each approach, we consider the following relevant features according to the different challenges confronted in this thesis:

- Regarding the challenge of involving users to develop or modify their own applications:
  - **User participation:** it indicates whether users, who are non-professional software developers, create or modify a software artifact at some point.
  - **System-aided:** it points out if the system aids users throughout the creation or modification of a software artifact and an example of how the approach achieves it. For instance, guiding users with wizards that show the steps that they may follow, automatic software creation, etc.
  - **Technique:** it indicates the technique that has been used to involve users.
- Regarding the challenge of involving users in modeling tasks:
  - **Model-based:** it indicates whether the approach is driven by models to specify the software artifact.
  - **Modeler experts vs users:** it shows who participates in the description of models (modeler experts, users or both).
- Regarding the challenge of achieving non-intrusive interoperability between models of heterogeneous modeling approaches:
  - **Different modeling approaches:** it shows whether the approach interchanges model descriptions with a different modeling approach.
  - **Interoperability mechanisms:** it describes if the interchange of information is performed among models of heterogeneous modeling approaches and how. For instance, an intermediate artifact

(such as a weaving model) is used to support the interchange of information.

- **Intrusive with MM:** it indicates whether the interoperability mechanisms require that the structure of models (Meta-Models) is modified in order to interchange information.
- **Collaborative modeling mechanisms:** it indicates if the approach provides a process where a number of people (modeler experts, users or a combination) actively contribute to the creation of a model.
- **Tool support:** it shows if the interoperability mechanisms are supported by tools, or by contrast, are presented at the theoretical level.

The information about each feature is outlined using the template of Table 3.1. In case that a feature is not supported or there is not published information, the X character will be shown. By contrast, if the feature is supported, the Y character will be shown. In addition, this table also shows the following features:

- **Application Domain:** it points out whether the approach has to be applied in some specific domain, or if it is domain independent.
- **Application Process:** it indicates if a process is provided for the application of the approach in different ones.
- **Limitations:** it summarizes the specific limitations of each approach.

As recommended by [101], manual and automated methods were used to make a selection of approaches in papers of leading journals and relevant conferences. The inclusion criteria was to: 1) be at least in one of the

EUD	User Participation	
	System-aided	
	Technique	
Modeling	Model-based	
	Modeler experts vs users	
Interoperability	Different modeling approaches	
	Interoperability mechanisms	
	Intrusive with MM	
	Collaborative M. Mechanisms	
	Tool support	
Application Domain		
Application Process		
Limitations		

Table 3.1: Template for showing the most relevant features of each approach

two categories of the classification criteria (involving users by following EUD techniques/metaphors and achieving MDD interoperability), and 2) present two or more features that are relevant to the different challenges confronted in this thesis.

The next two subsections describe the approaches found according to the identified categories and the features above explained. Each approach is summarized using the template.

3.2 Approaches for Involving Users

Since users are the "owners" of the problem, many approaches have arisen for involving them at some point of the development process. The majority of these approaches are focused on implementing an EUD technique or metaphor that were described in the previous chapter. Next, some relevant examples are presented.

Pervasive Interactive Programming (PiP) [34] follows the Programming by Example technique presented in the previous chapter. It employs a "show-me-by-example" approach allowing non-technical users to "program" their environment to suit their particular needs. PiP provides a platform that utilizes the physical user space as the programming environment. All the user needs to do, is simply to show the system the required functional behavior by demonstrating the required physical actions within the environment (see left side of Figure 3.1).

Figure 3.1: PiP pervasive environment and UI control panels

PiP has been inspired by the ease in which people perform daily routine tasks (eg. switching on the lights when a room gets dark, muting the TV sound when a telephone rings, etc). The approach finds a way of programming that is natural and imitates familiar practices as much as possible, without the need for the users to follow a set of rigid logical sequences of actions. The communication between PiP, the user and

the environment is via an eventing mechanism.

PiP is based on the concept of a MAp. A MAp contains a collection of rules that determine the behavior of the environment. Rules have two parts: the Antecedent (which are the conditions that enable the rule) and Consequent (which is the actions that are executed if the conditions are satisfied). In order to create a MAp, the user can use any of the following three methods: (1) physically interacting with the devices themselves by demonstrating the functionalities that the MAp should have via simple familiar interaction (e.g., by using a wall switch to turn on a light); (2) using a UI control panels (which are shown in the right side of Figure 3.1) that allows the user to "drag drop" device representations by engaging them in graphical activities; and (3) a combination of the above two methods. To terminate a MAp, the user simply clicks on the "stop" button of the interface. To execute a MAp, the user needs only to drag the MAp graphical representation and drop it into a "play" button located at the top of the User Interface (UI) control panels. To terminate a MAp the user simply clicks on the "stop" button.

Table 3.2 summarizes the features of this approach according to the presented template.

a CAPpella [30] is a Context-Aware Prototyping environment intended for end-users, which follows the technique Programming by Example. Users "program" their desired context-aware behavior (situation and associated action) in situ, without writing any code. However, this approach is limited to situations where a user can be reasonably expected to come up with a static, well-specified rule in a timely fashion that accurately describes the desired context-aware behavior.

A CAPpella uses a combination of machine learning and user input to support the building of context-aware applications through programming by example. Specifically, a user of a CAPpella demonstrates a context-aware behavior that includes both a situation and an associated action. In addition, an user interface is provided to indicate what portions of the demonstration are relevant to the behavior and trains a CAPpella on this behavior over time by giving multiple examples. Once trained, she can run a CAPpella, and it will enact the demonstrated

EUD	User Participation	Y
	System-aided	Y. Rules automatically generated
	Technique	Programming by Example
Modeling	Model-based	X
	Modeler experts vs users	X
Interoperability	Different modeling approaches	X
	Interoperability mechanisms	X
	Intrusive with MM	X
	Collaborative M. Mechanisms	X
	Tool support	X
Application Domain	Specific. Pervasive environment	
Application Process	X	
Limitations	-Users only can program basic event/action rules. -Low expressivity.	

Table 3.2: PiP. Summary of its most important features

behavior: performing the demonstrated action whenever it detects the demonstrated situation.

The user interface is divided into three parts. In the left frame, there is a video player that allows the user to view the recorded video and listen to the recorded audio. In the right frame at the top, the user can view events detected in the recorded sensor data, and on the bottom, the user can view actions that s/he took during the recorded session. After viewing the captured data, the user can annotate the data: selecting the streams of information she considers to be relevant to the behavior being created and the actions she wants a CAPpella to perform on her behalf. Moreover, the user sets a start and end time for all the streams to indicate when the behavior started and when it ended. Figure 3.2 shows a snapshot of the user interface being trained for a meeting. The user has selected a start and end time and deselected the location and RFID data streams. The actions shown are turning the lights on and off, and starting the notes recording program.

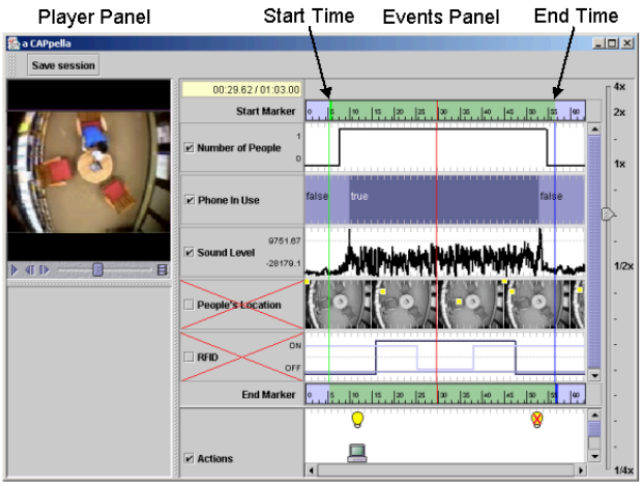


Figure 3.2: a CAPpella user interface

In order for a user to demonstrate a context-aware behavior, a CAPpella must have multimodal sensing capability to capture both the situation and the action that should be taken. a CAPpella currently uses an overhead video camera, a microphone, RFID antennas and tags, an instrumented computer (for login, logout, sending email, loading recently used files and for capturing user notes), and a switch that detects whether a phone is in use to capture events that occur during the demonstration of the situation in order to capture both *When the user starts the recording* (the sensors begin storing time-stamped data into separate logs) and *When the user stops the recording* system (the sensors stop sensing and event detection on the data logs begin). The user repeats this process a small number of times over a period of days or weeks and improves a CAPpella's ability to recognize this behavior with the new data. After a sufficient number of training examples have been provided, the user requests a CAPpella to recognize the situation, and when it does, it performs the demonstrated actions.

An application example using a meeting and medicine-taking scenario, Dey et al. illustrate in [30] how a user can demonstrate different behaviors to a CAPpella.

Table 3.3 summarizes the features of this approach according to the presented template.

Capture and Access Magnetic Poetry (CAMP) [38] is an end-user programming environment that allows users to create context-aware applications for home. CAMP has a user interface that is based on the Magnetic Poetry Metaphor (described in the previous chapter), which allows users to create applications in a way that takes advantage of the flexibility of natural language (see Figure 3.3).

EUD	User Participation	Y
	System-aided	Y. Ability to recognize the situation and perform actions.
	Technique	Programming by Example
Modeling	Model-based	X
	Modeler experts vs users	X
Interoperability	Different modeling approaches	X
	Interoperability mechanisms	X
	Intrusive with MM	X
	Collaborative M. Mechanisms	X
	Tool support	X
Application Domain	General	
Application Process	X	
Limitations	-Users have to train the system by recording events, which sometimes can be tedious for them. -The expressiveness is limited since the approach only supports the behavior that can be demonstrated.	

Table 3.3: CAPpella. Summary of its most important features

CAMP enables users to create programs that reflect the way they conceive of the desired application, rather than requiring that users specify applications in terms of devices. From users' magnetic poetry-based application descriptions, CAMP generates a specification of a valid capture application that can be executed in a capture-enabled home environment. CAMP makes use of a restricted and domain-

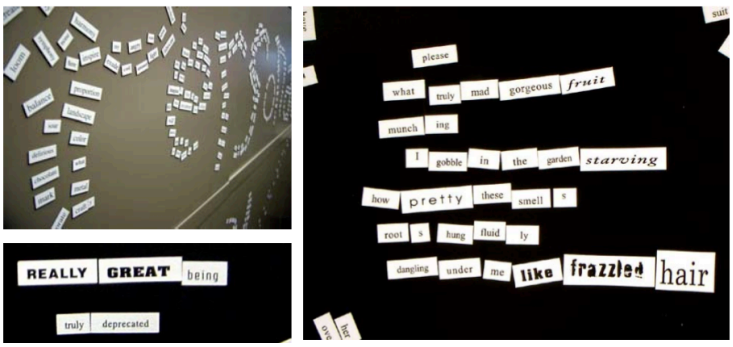


Figure 3.3: Examples of different magnetic poetry arrangements

specific vocabulary, it avoids many of the difficulties involved in parsing natural language.

CAMP serves as an interface to INCA [102], an infrastructure that provides abstractions for the development of capture and access applications. The interface is designed to allow people to use an input language with which they are comfortable and that lets them express their ideas flexibly; CAMP automatically generates the technology-oriented application specifications necessary for realizing the applications. By doing so, CAMP allows non-developers to create programs that are valid ubicomp applications without having specialized programming knowledge. The constrained vocabulary makes clear to users what their choices are, and what aspects of the system they can play with or configure.

The users use four w's word categories (*who, what, where, when*) to capture and access in describing applications. Some examples of each are there:

- who: I, me, everyone, no one, family, stranger, baby, wife, Billy, etc.
- what: picture, audio, video, conversation, etc.

- where: kitchen, living room, home, everywhere, etc.
- when: always, later, never, a.m., morning, day, week, month, before, hour, minute, Sunday, January, once, now, every time, etc.
- general: 1, 2, a, the, record, remember, view, save, keep, microphone, speaker, etc.

Figure 3.4 shows an snapshot of the CAMP interface while a sentence is composed. Once the end-user has composed a sentence, the system automatically translates it into instructions and parameters for devices, using a custom dictionary to reword and restructure the user's terms into a format that can be parsed. This translation is displayed in the bottom frame of the interface as feedback to the user. The INCA infrastructure abstracts the lower level details involved in the development of capture and access applications, and provides customizable building blocks that support interfaces for capturing and accessing information, components for storing information, a way to integrate relevant streams of information, and the removal of unwanted data.

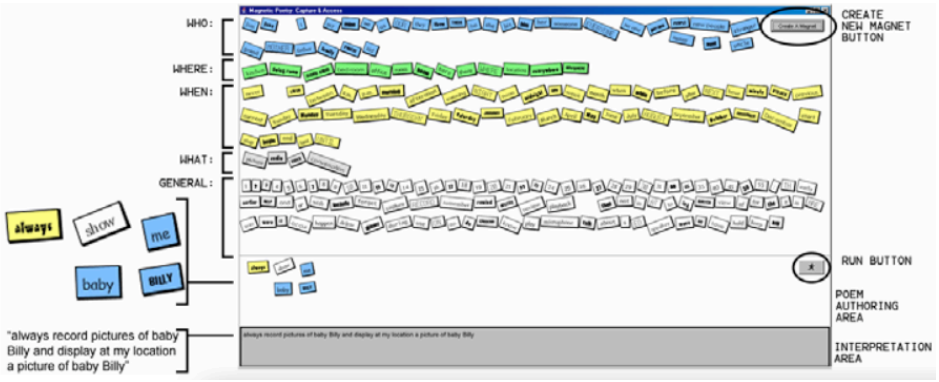


Figure 3.4: The Capture & Access Magnetic Poetry interface

Table 3.4 summarizes the features of this approach according to the

presented template.

EUD	User Participation	Y
	System-aided	Y. Automatic translation of sentences into instructions.
	Technique	Magnetic Poetry Metaphor
Modeling	Model-based	X
	Modeler experts vs users	X
Interoperability	Different modeling approaches	X
	Interoperability mechanisms	X
	Intrusive with MM	X
	Collaborative M. Mechanisms	X
	Tool support	X
Application Domain	General	
Application Process	X	
Limitations	Expressiveness is limited for applications that capture behavior using devices within the physical environment (such as cameras and interactive displays).	

Table 3.4: CAMP. Summary of its most important features

The Accord Toolkit [12] enables people to easily administer and reconfigure services based on embedded devices around the home by means of the Tangible Toolbox. This toolbox also enables devices to be integrated with each other through several different editors.

The conceptual model of the developed Tangible Toolbox is made up

of services that compose components. These components are seen as three different kinds of *transformers*, two that either transform physical properties to digital data or vice versa and the third kind that takes some digital data and transforms into another form of digital data. The transformers are the fundamental building blocks of the Tangible Toolbox. The underlying infrastructure of the toolbox is based on the model and a set of editors addressing different user groups and needs. These editors are:

The Graph Editor is directed to expert users such as programmers and displays all components in a graph. This editor gives a good overview of all existing components and how they are connected. The other editors are aimed at the inhabitants of households. With the Linker Device users can explore what properties a physical device in the home expose and through the Linker Device link these with properties of other physical devices (see Figure 3.5.a).

The Puzzle Editor is a graphical interface to compose services from components. This editor is based on the jigsaw metaphor (described in the previous chapter) to enable a user the connection of components through a series of left-to-right couplings of puzzle pieces (see Figure 3.5.b).

The Paper Puzzle Editor utilizes paper based identification technology. This editor aims at creating an interface to the Tangible toolbox that is not perceived as computer interface. Each component is represented as a physical puzzle piece and in the same way as in the graphical Puzzle Editor; a service is created through connecting these pieces in a left-to-right order (see Figure 3.5.c).

As Figure 3.5 shows, the Puzzle Editor is focused on enabling users the construction of assemblies, it is described further. This editor discovers a local dataspace where transformers are registered. In order to make itself available for use a transformer exports itself to the distributed dataspace. The transformer is introspected and the properties associated with it are made available as input and output points, each transformer also has a jigsaw piece property which exports how it should appear in the editor. Each room has a dataspace associated with it and components that can be accessed from that room are registered with the dataspace.

The Puzzle editor is composed of two distinct panels: a list of available components (shown as jigsaw pieces, see the top of Figure 3.5.b) and an editing canvas (see the bottom of Figure 3.5.b). Jigsaw pieces can be dragged and dropped into the editing canvas or workspace. When a jigsaw piece is dragged onto the workspace it clones itself and becomes a symbolic link to the underlying component it represents. The editing canvas serves as the work area for connecting pieces together and visualizing their activities. When properties related to jigsaw pieces in the dataspace are updated, the corresponding jigsaw piece changes its color and a short audio clip is played.

Table 3.5 summarizes the features of this approach according to the presented template.

Alfred [11] follows the butler metaphor described in the previous chapter through the use of a macro programming approach, which enables a user to compose a program via Programming by Example (using verbal or physical interactions). Alfred is an end-user programming interface that allows a user to "program" the system by telling it the name of a

(a) The Graph Editor

(b) The Puzzle Editor

(c) The Paper Puzzle Editor

Figure 3.5: Accord Toolkit editors

new goal, demonstrating one or more plans for achieving that goal, and finally telling the system the conditions under which it would prefer one plan to another. Similarly, the user can name events that arise in the environment and tell the system what goals should be posted when those events arise. Each of these steps can be done by simple verbal commands or other natural forms of interaction.

Alfred works with *Rascal* [103] and *ReBa* [104] two systems, which are responsible for the adaptive and reactive components. On the one hand, Rascal is able to support interactions in a variety of spaces with very different capabilities. Rascal provides a crucial layer of abstraction by allowing applications to make high-level service requests, such as delivering a message to the user. Rascal then evaluates all available methods for satisfying the request, effectively producing a plan that takes into account the availability of the hardware and software resources in the current environment. Additionally, Rascal can take "advice" from other agents on what kinds of resources are preferable in what context. For example, an agent detecting activity context through ReBa will discourage the use of audio devices in favor of displays when

EUD	User Participation	Y
	System-aided	Y. It indicates which pieces are available for target connections.
	Technique	Jigsaw metaphor
Modeling	Model-based	X
	Modeler experts vs users	X
Interoperability	Different modeling approaches	X
	Interoperability mechanisms	X
	Intrusive with MM	X
	Collaborative M. Mechanisms	X
	Tool support	X
Application Domain	Specific: smart home systems	
Application Process	X	
Limitations	Expressiveness is limited to the pieces and the rules that define which ones can be connected.	

Table 3.5: The Accord toolkit. Summary of its most important features

the user is talking on the phone. This layer of abstraction enables the design of new applications without having detailed knowledge of the environments they would be running in. This software has been running in several offices, a conference room, a living room, and a bedroom.

ReBa automatically reacts to some of the events taking place within its boundaries and context. For example, an environment should illuminate the room upon a person's entry but it should not illuminate the room

upon a person's entry if the room is already occupied by people. The core component of ReBa is its *Behavior Coordinator*, which resolves the potential conflicts and dependencies among the individual behaviors. Although all behavior bundles are written by software engineers, it is the responsibility of the owner of any individual space to choose the most appropriate combination of behaviors for his space.

Thus, Alfred is essentially a multi-modal macro recorder. Upon a user's request, the system begins recording all of his actions, primarily spoken commands. When the recording is done, Alfred assigns one or more spoken names to the recorded sequence. Alfred can also add hardware triggers to it. The recorded macros are simple task sequences lacking explicit conditionals. Macros can, however, call other macros, giving users the capability to create abstractions. Interacting with Alfred is a sequence of tasks forming a procedure, which is quite familiar to most users from recipes and other instructions.

Table 3.6 summarizes the features of this approach according to the presented template.

The user can perform using Alfred the following: 1) Recording a New Macro, 2) Adding a Hardware Trigger and 3) Invoking the tasks. An example to create a new trigger sequence is:

User: *When I press this button [user presses one of the free buttons] run the "Good morning, computer" sequence.*

Computer: *Please press the button again for confirmation.*

[User presses the button again]

Computer: *Done!*

After, if somebody presses the button, the "Good morning, computer" macro will be automatically executed. A recorded task sequence can be

EUD	User Participation	Y
	System-aided	Y. The system automatically creates tasks from users' recordings and tells the users if some tasks have failed.
	Technique	Butler metaphor Programming by Example
Modeling	Model-based	X
	Modeler experts vs users	X
Interoperability	Different modeling approaches	X
	Interoperability mechanisms	X
	Intrusive with MM	X
	Collaborative M. Mechanisms	X
	Tool support	X
Application Domain	General	
Application Process	x	
Limitations	-Limited expressivity due to the lack of conditionals. -The user has to be familiar with vocabulary to record and invoke the tasks.	

Table 3.6: Alfred. Summary of its most important features

invoked in three different ways: through a spoken command, through a hardware trigger (if defined), and through a graphical user interface (if present).

The Spreadsheet Paradigm [105] follows a Natural Programming technique since spreadsheet languages are widely used End-user program-

ming languages. Burnett et al. [106, 107] have prototyped their approach in the spreadsheet paradigm. Their prototype includes the following:

- An interactive testing methodology to help end-user programmers test. To do this, one of the components is the "What You See Is What You Test" (WYSIWYT) methodology for testing [108]. WYSIWYT allows users to incrementally edit, test and debug their formulas as their programs evolve, visually calling users' attention to untested cells by painting their cell borders in red (see the red color in Figure 3.6) meanwhile tested cells are painted blue.

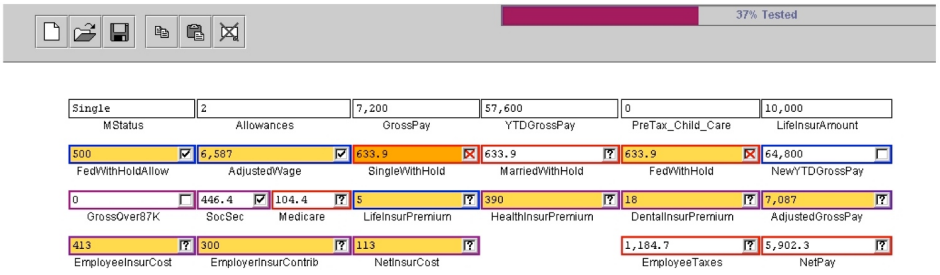


Figure 3.6: An spreadsheet paradigm example that provides feedback using colors

- Fault localization capabilities to help users find the faults that testing may have revealed. To do this, the "Help Me Test" (HMT) feature [109] suggests test values for user-selected cells or user-selected dataflow arrows.
- Interactive assertions to continually monitor values the program produces, and alert users to potential discrepancies.

Figure 3.6 shows an example of spreadsheet paradigm environment in which the fault-localization feedback of the system shades the cells that

have a higher likelihood of faults, compared to lighter shades for cells that are less likely to contain faults.

Table 3.7 summarizes the features of this approach according to the presented template.

EUD	User Participation	Y
	System-aided	Y. Inmediate feedback using colors.
	Technique	Natural Programming
Modeling	Model-based	X
	Modeler experts vs users	X
Interoperability	Different modeling approaches	X
	Interoperability mechanisms	X
	Intrusive with MM	X
	Collaborative M. Mechanisms	X
	Tool support	X
Application Domain	General	
Application Process	X	
Limitations	-Lack of data abstraction features. -Simple model input/output that consists in the ability to enter constant formulas.	

Table 3.7: The spreadsheet paradigm. Summary of its most important features

The Whyline [36] is a debugging interface for asking questions about program behavior that follows the Natural Programming technique described in the previous chapter. It uses questions because a debugging

activity always begins with a question, and programmers/users must answer their question using existing tools and their limited capabilities. Hence, this approach proposes to remove this hurdle by allowing programmers/users to directly ask the questions they naturally want to ask.

The Whyline is prototyped in Alice, the environment shown in Figure 3.7 as well as: (1) the object list, (2) The 3D world view, (3) the event list, (4) the currently selected object's properties, methods, and questions, and (5) the code area. Alice is an event-based language that simplifies the creation of interactive 3D worlds. Code is created by dragging and dropping tiles to the code area and choosing parameters from popup menus. This interaction prevents all type and syntax errors.

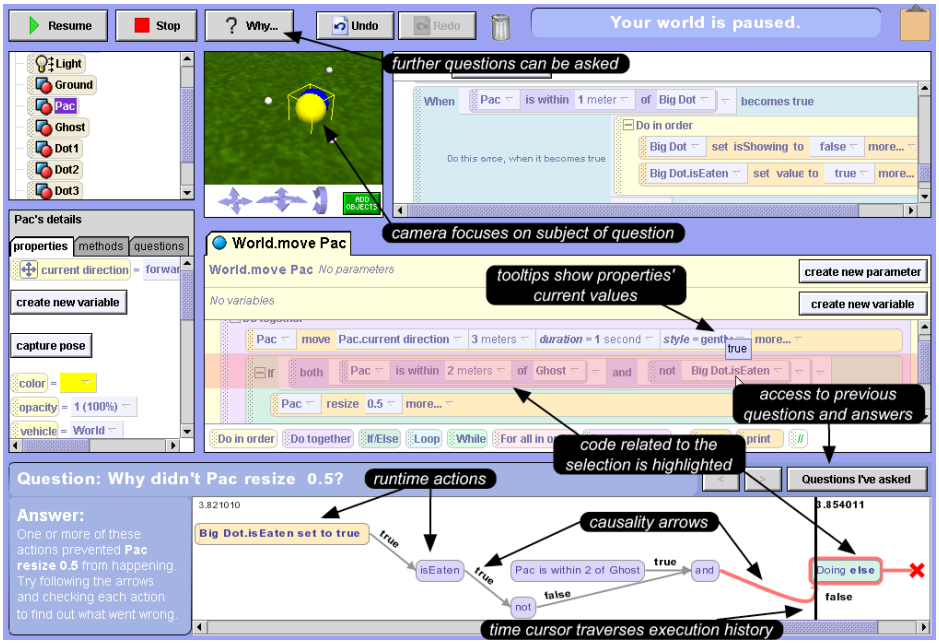


Figure 3.7: The Alice programming environment before the world has been played

Table 3.8 summarizes the features of this approach according to the

presented template.

EUD	User Participation	Y
	System-aided	Y. Visual highlighting helps with diagnosis and repair activities.
	Technique	Natural Programming
Modeling	Model-based	X
	Modeler experts vs users	X
Interoperability	Different modeling approaches	X
	Interoperability mechanisms	X
	Intrusive with MM	X
	Collaborative M. Mechanisms	X
	Tool support	X
Application Domain	General	
Application Process	X	
Limitations	Questions are predefined (the user is limited to explore the available questions and ask).	

Table 3.8: The Whyline. Summary of its most important features

The Whyline answers the question by analyzing the runtime actions that did and did not happen, and provides an answer as Figure 3.7 shows. The Whyline supports observation and hypothesizing by increasing the visibility of the actions that likely contain the fault. The arrows represent data and control flow causality, which are labeled by the action they point to. The arrows help the user to follow the runtime system's

computation and control flow. The user interacts with the timeline by dragging the time cursor (the vertical black line in Figure 3.7). In case that the user moves the cursor over an action, the action and the code that caused it become selected, supporting diagnosis and repair. The most helpful feature of the Whyline seems to be the question menu and visual highlighting that helps with diagnosis and repair activities.

The above approaches serve to analyze different ways of lowering barriers to users in works, which are focused on implementing an EUD technique or metaphor but they are not model-based. In the majority of model-based approaches, closer languages such as DSLs or DSVLs, different views, or abstraction levels are provided in order to make users' participation easier in a concrete domain. Next, two model-based approaches are presented as example.

PANTO [110] provides a natural language interface for executing queries that acquire information from ontologies. Ontology refers to a knowledge base that includes concepts, relations, instances that together model a domain for storing a lot of knowledge. Thus, this approach bridges the gap between basic semantic web and real-world users since users play a key role in semantic web sharing and exchanging information but they may acquire formal knowledge in ontologies to obtain their needed information (e.g., the ontology syntax, some formal query language, and the structure and vocabulary of the target ontology).

PANTO follows several steps to translate natural language queries to SPARQL queries since SPARQL has been recommended as the standard query language for the semantic web community. These steps are summarized as follows:

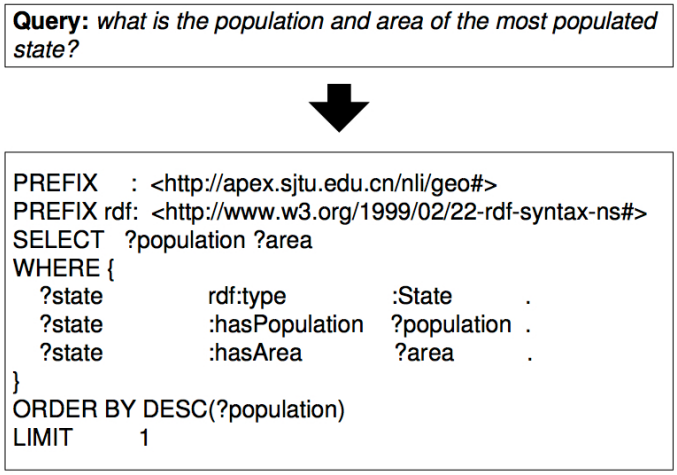


Figure 3.8: Panto example of translating natural language queries to SPARQL queries

1. It translates Natural Language queries to entities (concepts, instances or relations). This step is mainly supported by the *Lexicon*, which is composed of the following components:

   - Ontology Entities are extracted and stored for fast access and matching. In particular, ontology entities and their names are put into a special hash table, in which a key maps to a set of ontology entities and an ontology entity can be obtained by different keys. Given a word from the natural language query, the Lexicon will acquire a set of possible entities. Proper nouns are also extracted from the ontology for fast access and matching.
   - General Dictionaries bridge the gap between user vocabulary and ontology vocabulary. Thus, user's query concepts can be matched to the ontology concepts. Moreover, the general dictionaries enable Panto to translate some user's words such as "how long..." in properties such as 'length".

- User-Defined synonyms allows users to optionally define their own "synonymy words" since users may use jargons and abbreviations to denote entities, words from general dictionaries only may not be enough. Thus, a set of words that match the same entity in the ontology can be defined.

2. It extracts the entities in the parse trees as pairs to form an intermediate representation called *QueryTriples*. Then, PANTO maps *QueryTriples* to *OntoTriples* which are represented with entities in the ontology. Finally, *OntoTriples* are interpreted as SPARQL. To do this, the *Translator* is the backbone of PANTO, which carries out the deep parser of parse trees. Thus, the words of natural language queries store facts of the domain model that are stated in the triple form *<subject, predicate, object*. The subject and the object may be classes, instances or literal values and usually should be named with words or phrases. The predicate may be prepositions, verbs, verb phrases and so on, and sometimes may also be phrases.

Table 3.9 summarizes the relevant information of PANTO according to the presented template.

Although PANTO address some complex sentences in natural language such as negations or comparative, it has some limitations since the ambiguity and complexity make difficult for a machine to understand arbitrary natural language (such as queries involving count on instances) and more limitations arise when more features are added to SPARQL. In addition, PANTO is focused on the translation steps, so it has a weak spot in user interaction since PANTO does not guide users to express their natural language queries.

BaVeL [65] is a DSVL to visually specify rules for the syntactic and semantic validation of different models for presenting the result in an intuitive way to users. This is performed by translating the analysis results in terms the user can interpret. As a consequence, the analysis results may not be ignored or misinterpreted by users, which may avoid wrong design decisions.

EUD	User Participation	Y
	System-aided	X
	Technique	Natural Programming
Modeling	Model-based	Y
	Modeler experts vs users	Modeler experts
Interoperability	Different modeling approaches	X
	Interoperability mechanisms	X
	Intrusive with MM	X
	Collaborative M. Mechanisms	X
	Tool support	X
Application Domain	SPARQL queries	
Application Process	X	
Limitations	-Weakness in user interaction (users are not guided to correctly express their natural language queries). -PANTO translators only support SPARQL queries and they cannot be reused in other approaches.	

Table 3.9: PANTO approach. Summary of its most important features

Specifically, the DSVL allows the modeler expert to select input data, filter the system models, perform the verification by calling external or internal analysis tools, and select the output format of the validation. The latter includes how results should be reflected in the source model. Once the previous steps have been performed, a customized modeling environment for the DSVL is automatically generated. Such environment allows the user to build models as well as verify the properties that the modeler expert made available.

Figure 3.9 shows the DSVL generated interface, which includes one button for each defined validation property, executes the analysis and returns the results. In particular, the Figure 3.9 shows the result obtained after executing the analysis called *state reachability* (i.e. the input to the analysis was modeled as a graph element). The user selected such analysis by clicking on the corresponding validation property.

This process is transparent to the user who just selects the property to analyze, and the verification results are returned in a proper way to the user (e.g. in terms of the original language) as defined by the modeler expert through BaVeL. Moreover, Guerra et al. states that BaVeL is customizable to any source DSVL.

Table 3.10 summarizes the relevant information of BaVeL according to the presented template.

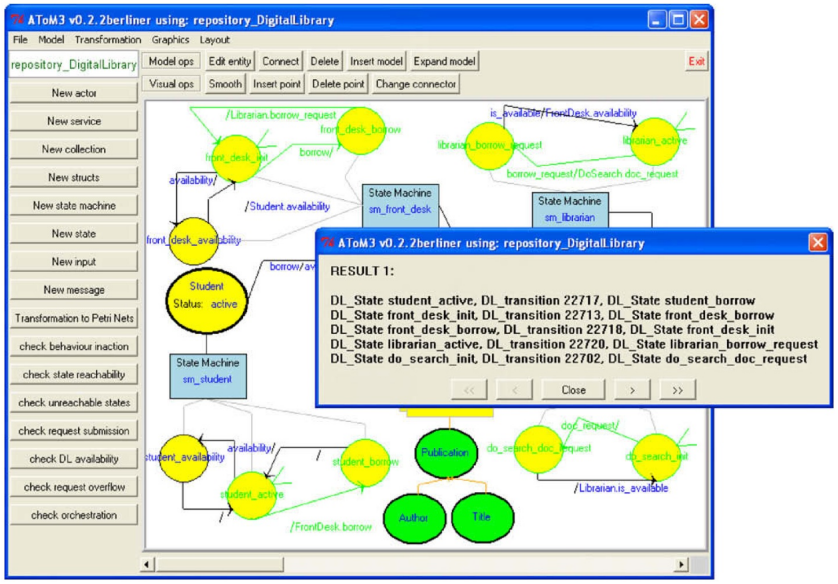


Figure 3.9: Snapshot of BaVeL implementation. Validation mechanisms in terms users can interpret

3.3 Approaches for Achieving MDD Interoperability

Interoperability mechanisms could provide benefits at the different stages of the software development in MDD approaches. For instance, some MDD approaches were carefully designed to interoperate from a different MDD approach in order to provide a different language to edit models. Other approaches interoperate to integrate different development groups in a common model development project.

Therefore, it is possible to find a variety of MDD approaches that carry out interoperability in order to exchange model information in literature. Next, some relevant examples of model-based approaches that achieve MDD interoperability are presented and the relevant information for each one is shown using the table template previously presented.

EUD	User Participation	Y
	System-aided	Y. Selection mechanisms and results are shown in a language closer to users
	Technique	X
Modeling	Model-based	Y
	Modeler experts vs users	Users
Interoperability	Different modeling approaches	X
	Interoperability mechanisms	Y. Weaving model to support model-to-model transformations within the same modeling approach
	Intrusive with MM	Y
	Collaborative M. Mechanisms	X
	Tool support	Y
Application Domain	General	
Application Process	Y	
Limitations	The modeler specifies the properties to analyze in models and the user just builds models and selects the properties to analyze	

Table 3.10: BaVeL approach. Summary of its most important features

Voelter and Solomatov [111] advocate the integration of different languages by language modularization and composition. Modular Languages use a relatively small general-purpose core and can be extended with more (domain specific) concepts as needed. Thus, each language module may address a specific concern of software development in general, of a specific system or platform of a business domain. A

language module is similar a traditional framework or library, but it comes with its own syntax, IDE support, type system, and compiler or transformation engine.

In modular languages, the use of both DSLs and code generation is an important argument since DSLs are closely aligned with a business domain or a specific concern of a software system (such as persistent data definition, workflow or component structures). DSLs can be put into module libraries and can be used directly, or extended slightly to tailor them to a specific architecture. It is certainly not possible to define all these language modules completely independent of each other, so a clear layer structure between the modules is necessary in order to allow language modules to work with each other. In particular, this approach explains techniques for language extension and composition based on projectional editors in general, and JetBrains' Meta Programming System [112](MPS) specifically.

Figure 3.10 shows an example of this approach that embeds independent languages. Specifically, it embeds SQL into Java and it is implemented using MPS.

```
public void aMethod(string p) {
  SELECT * FROM myTable WHERE x == p
}
```

Figure 3.10: A select SQL statement embedded in Java

To define a language extension with MPS, several steps have to be performed. These steps are summarized as follows:

1. Definition of the structure of the language A (abstract syntax) and the structure of the extended concepts of the language B.

2. Definition of the representation of concepts in the model editor (concrete syntax).
3. Definition and validation of variable types by creating rules.
4. Editor Tuning to integrate the extended concepts in editors. Defining an usable editor that not only adds the extended concepts by selecting them in a menu but also recognizes them by simply typing in the editor, is an additional effort that modeler experts have to carry out (e.g., as the previous example shown in Figure 3.10 that integrates SQL in a Java method and references a method parameter by just typing the code).
5. Definition of transformations to generate and compile the code. In case of DSLs or domain-specific extensions of general purpose languages cannot be directly executed. The model has to be translated into a language for which some kind of execution infrastructure (a compiler or interpreter) exists. In an environment where models and programs are treated the same in that they are both stored as an abstract syntax tree and projected for editing, there are two different scenarios for code generation:
   - DSLs or language extensions typically need to be mapped to general purpose languages such as Java or C.
   - Since the general purpose languages themselves are represented via an Abstract Syntax Tree and projection, the programs cannot be feed directly to the compiler, a text representation has to be generated from them.

Table 3.11 summarizes the relevant information of this approach according to the presented template.

At this point, it is also worth pointing out that modular languages

EUD	User Participation	X
	System-aided	X
	Technique	X
Modeling	Model-based	Y
	Modeler experts vs users	Modeler experts
Interoperability	Different modeling approaches	Y
	Interoperability mechanisms	X
	Intrusive with MM	Y
	Collaborative M. Mechanisms	X
	Tool support	Y
Application Domain	General	
Application Process	Y	
Limitations	-Existing languages have to be re-implemented in a specific projectional workbech. -Users have to be experts in concepts of different languages to carry out the system description.	

Table 3.11: Voelter and Solomatov approach. Summary of its most important features

are flexible because the concepts involved can be chosen, and they are convenient for users who use concepts from different languages within the same tool. Nonetheless, users have to be experts in every concept of each integrated language, which could make system descriptions for non-expert users difficult.

Giachetti [113] propose a process that integrates UML and DSL models in an unique Model-Driven Development solution. This process is comprised of the following 4 steps:

1. Definition of meta-models of the involved modeling approaches (source and target modeling approaches).

2. Definition of the integration meta-model to identify the equivalences between the meta-models involved and to fix the mapping issues that are produced by structural differences that may exist. This integration meta-model is also called *pivot meta-model*. The differences between a pivot meta-model and a weaving meta-model are related to their definition and use. A weaving meta-model is instantiated to represent links among constructs of the involved meta-models. By contrast, a pivot meta-model can be a pre-defined representation of concepts for the interoperability domain, or can be generated from the meta-models of the modeling approaches that must interoperate.

   Moreover, this approach defines constraints for the definition and generation of the pivot meta-model such as: 1) *all the classes from the pivot meta-model must be mapped*, 2) *the mapping is defined between elements of the same type*, and 3) *an element from the integration meta-model is only mapped to one element of the target meta-model*. The violation of these rules may require the modification of the meta-models in order to fulfill them.

3. Automatic UML Profile Generation. This step considers the automatic generation of the UML profile that implements the meta-model extensions that are required to customize the abstract syntax of a target modeling language with the modeling informa-

tion of the MDD approach involved.

4. Generation of Model-Interchange Mechanisms considers the generation of the necessary model transformations to automatically obtain from the models, which are defined with the customized modeling language appropriate inputs (models), to specific MDD tools such as model compilers.

Table 3.12 summarizes the relevant information of this approach according to the presented template.

Figure 3.11 shows an example of this approach in which specific modeling features of a DSL have been integrated into UML. In particular, the example shows that UML has been extended with 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.

Guerra et al. [114] approach consists of a pattern-based approach for defining bidirectional relations (a weaving model) among modeling approaches. The main contribution of this proposal is an unique framework for the definition of specific inter-modeling patterns. These patterns also provide advantages such as identification of interoperability conflicts, and the generation of model-to-model transformations. Table 3.13 summarizes the relevant information of this approach according to the presented template.

Figure 3.12 provides the general scheme of this approach. In Step 1, the

EUD	User Participation	X
	System-aided	X
	Technique	X
Modeling	Model-based	Y
	Modeler experts vs users	Modeler experts
Interoperability	Different modeling approaches	Y
	Interoperability mechanisms	Y. Pivot model
	Intrusive with MM	Y
	Collaborative M. Mechanisms	X
	Tool support	Y
Application Domain	General	
Application Process	Y	
Limitations	Interoperability mechanisms are designed to interoperate between a DSL and UML.	

Table 3.12: Giachetti approach. Summary of its most important features

designer (who is a modeler expert) builds the "inter-model specification" using a pattern language, which can be analyzed in Step 2 (e.g., conflicts of patterns with respect to the language meta-models such as a pattern requires two links stemming from an object but the meta-model cardinality constraints only allow one). In Step 3, the designer chooses the usage scenario for the specification: transformation, model matching or model traceability. In the transformation scenario, the designer can decide whether the operational mechanisms are for forward

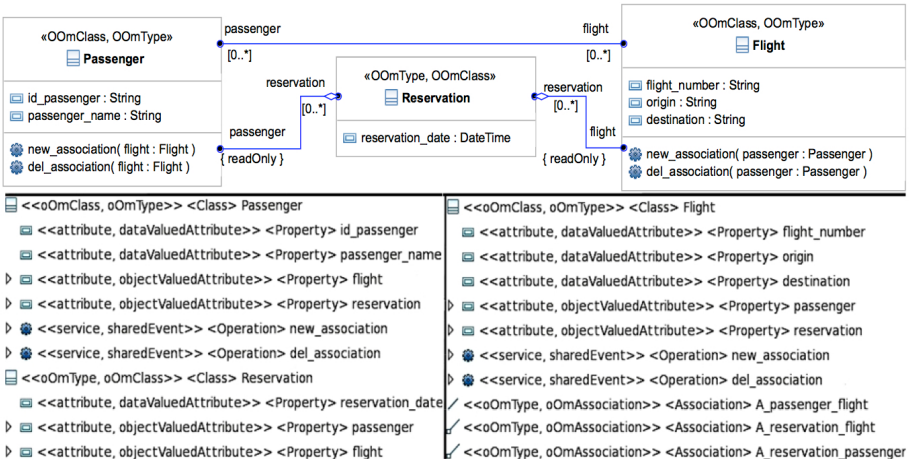


Figure 3.11: Example UML model extended

or backward transformation (the patterns are direction-independent and can be interpreted both ways). In this case the synthesized mechanisms will create a target model from a source one from scratch (forwards) or vice-versa (backwards). For model matching and model traceability, the generated operational mechanisms are able to create appropriate traces between the compared models, as well as to delete incorrect traces.

Klar et al. [21] shows how the MDD interoperability can be used to support a complete development process. In particular, this proposal is focused on the integration of requirement modeling into the MDD process.

Figure 3.13 shows a running example of this approach, which seeks to integrate descriptions of use case diagrams. These use case diagrams are described using different modeling languages and tools.

On the one hand, a standard requirements engineering tool is employed to create more detailed textual use case description and to record other sorts of (non-functional requirements) as the left side of Figure 3.13 shows. On the other hand, a UML tool supports the creation of high-

EUD	User Participation	X
	System-aided	X
	Technique	X
Modeling	Model-based	Y
	Modeler experts vs users	Modeler experts
Interoperability	Different modeling approaches	Y
	Interoperability mechanisms	Y. Weaving model
	Intrusive with MM	X
	Collaborative M. Mechanisms	X
	Tool support	Y
Application Domain	General	
Application Process	Y	
Limitations		

Table 3.13: Guerra et al. approach. Summary of its most important features

level use case diagrams as the right side of Figure 3.13 shows.

To integrate model descriptions, interoperability mechanisms have to be defined by a modeler expert. In particular, the modeler expert defines a set of mappings and transformation rules using both meta-models as input. Following the use case diagrams example, once the interoperability mechanisms have been defined, a modeler expert can describe use case diagrams (in either of the above mentioned tools) and interoperate between them as follows: (1) create a first set of use

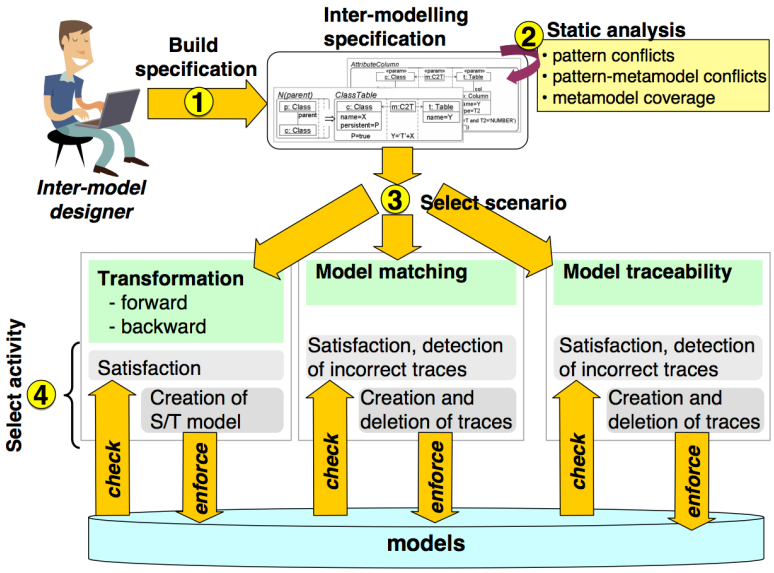


Figure 3.12: Guerra et al. interoperability general scheme

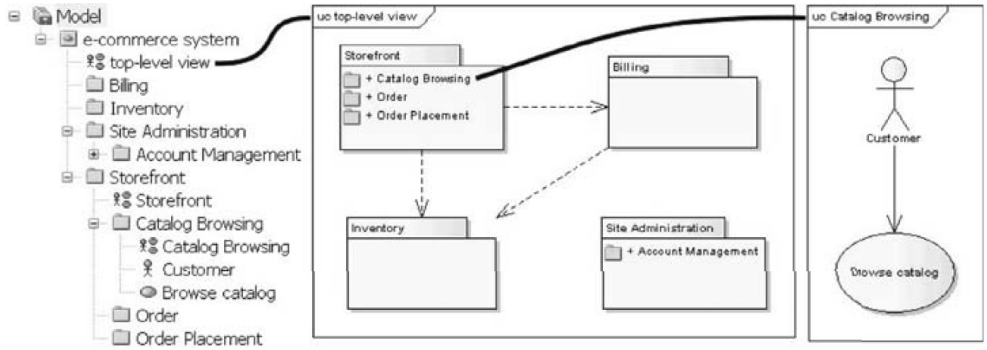


Figure 3.13: Interoperability between two tools that describe use case diagrams

case diagrams and translate them into skeletons of more detailed textual descriptions, (2) complete and modify the generated text skeletons, and (3) synchronize the result text description with the use case diagrams of (1).

Table 3.14 summarizes the relevant information of this approach according to the presented template.

EUD	User Participation	X
	System-aided	X
	Technique	X
Modeling	Model-based	Y
	Modeler experts vs users	Modeler experts
Interoperability	Different modeling approaches	Y
	Interoperability mechanisms	Y. Weaving model
	Intrusive with MM	X
	Collaborative M. Mechanisms	X
	Tool support	Y
Application Domain	Specific. Requirements Engineering	
Application Process	Y	
Limitations	Tools does not support *de facto* standard in the meta-modeling community, which makes the interoperability with existing modeling approaches difficult.	

Table 3.14: Klar et al. approach. Summary of its most important features

Kappel et al. [115] propose a framework for model-based tool integration which is based on conceptual modeling techniques (a weaving model and model transformations). This framework enables the design of integration models on a conceptual level in terms of UML component diagrams. Furthermore, this approach addresses recurring integration problems such as structural meta-model heterogeneities by means of reusable integration components.

In this approach, the weaving model is defined using a mapping language called CAR. This language provides nine different core mapping operators. These nine mapping operators result from the possible combinations between the core concepts of meta-metamodels, namely class, attribute, and reference, which also led to the name of the CAR mapping language. These mapping operators are designed to be bi-directional.

One important requirement for the CAR mapping language is that it should be possible to reconstruct the source models from the generated target models, i.e., any loss of information during transformation should be prevented.

Although this proposal reduces the modeler experts' effort by reducing the number of model elements of the manually created mapping model compared to the number of elements needed for the corresponding model transformations of other tools, the modeler experts' effort is increased if they have to integrate existing weaving models and transformation rules in the *de facto* model transformation standard into the CAR mapping language.

Table 3.15 summarizes the relevant information of this proposal according to the presented template.

Invar [16] advocates an integrative approach called Invar (INtegrated view on VARiability) that provides an unified perspective to users configuring products in multi product line environments, regardless of the different modeling methods and tools used internally. Thus, this approach facilitates the integration of variability models (such as feature models) by presenting the configuration options of multiple variability models created with different heterogeneous modeling approaches to the end-

EUD	User Participation	X
	System-aided	X
	Technique	X
Modeling	Model-based	Y
	Modeler experts vs users	Modeler experts
Interoperability	Different modeling approaches	Y
	Interoperability mechanisms	Y. Weaving Model in terms of UML component diagrams
	Intrusive with MM	X
	Collaborative M. Mechanisms	X
	Tool support	Y
Application Domain	General	
Application Process	Y	
Limitations	The weaving model is defined using the CAR mapping language, which increases the modeler experts' effort if interoperability is necessary with *de facto* standard modeling tools or vice-versa.	

Table 3.15: Kappel et al. proposal. Summary of its most important features

user in an integrated fashion.

Figure 3.14 shows the current state of practice that no integration tools are used and Invar. On the one hand, the current state of practice multiple heterogeneous variability modeling approaches are used by different organizations and there is no integration of the diverse tools supporting different notations. On the other hand, Invar approach

allows to "plug-and-play" variability models. "Plugging" refers to simply adding new variability models to a shared repository. "Playing" refers to presenting the options provided by variability models to end-users configuring a product. For this purpose, a variability model is seen as an autonomous entity, which can be plugged into the configuration space to provide configuration options. Autonomous however does not necessarily mean independent, because variability models may be related with each other. In particular, this approach allows using variability models distributed across multiple repositories by accessing them through Web Services providing configuration choices. An end-user works with a front-end for product configuration and can use the services without knowing details about the concrete variability models "behind" the services.

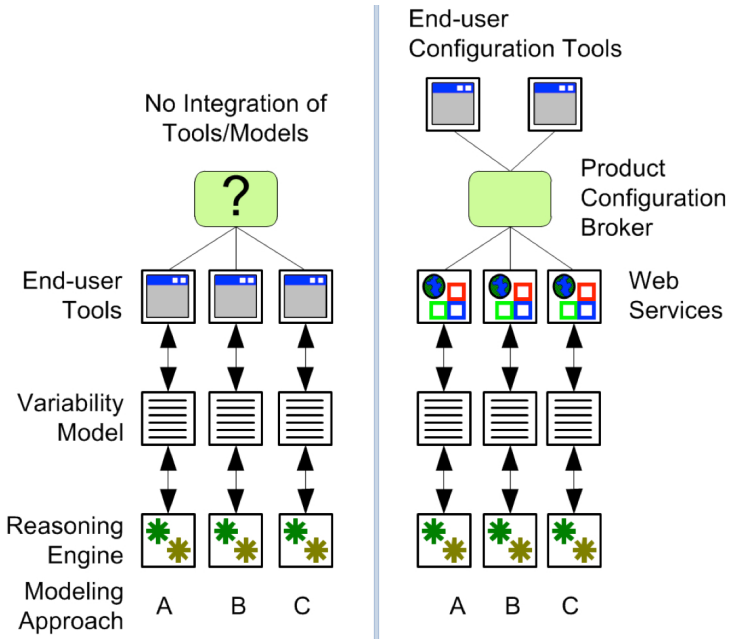


Figure 3.14: Configuration of multi product lines: no integration tools (left) and Invar approach (right)

As Figure 3.14 shows, there is a component in the Invar architecture that enables the communication between the Web services called *Configuration Broker*. The Configuration Broker enables the communication between the Web Services. It reads the inter-model dependency information to determine which Web Services are affected when products are configured. The configuration broker also translates events from the end-user configuring and passes them on to the Web Services that need to react to the end-user's interactions, which the Invar framework supports with a set of operations that manage information of variability models (such as get the selected features).

The end-user product configuration front-end can be a website or a stand-alone application and presents the choices defined in the variability models. End-user can select among the choices by answering questions such as "Do you want to enable international bank transfers?" in a natural way (like the Whyline approach previously described).

Table 3.16 summarizes the relevant information of Invar according to the presented template.

EUD	User Participation	Y
	System-aided	X
	Technique	Questions/answers to configure the variability
Modeling	Model-based	Y
	Modeler experts vs users	Both
Interoperability	Different modeling approaches	Y
	Interoperability mechanisms	Y. Controller based on operations and rules
	Intrusive with MM	X
	Collaborative M. Mechanisms	X
	Tool support	Y
Application Domain	Specific. Variability modeling	
Application Process	Y	
Limitations	-Interoperability mechanisms are focused on supporting variability models. -Users configure products by answering questions that activate/deactivate features in variability models (they do not participate in the creation of variability models).	

Table 3.16: Invar approach. Summary of its most important features

3.4 Discussion and Conclusions

This chapter has presented the state of the art by analyzing existing approaches and classifying them in two categories: approaches for involving users in the creation or modification of a software artifact, and approaches for

achieving interoperability between modeling approaches. For each approach, relevant features are analyzed as well as its limitations.

Table 3.17 shows the analyzed approaches and a summary of the obtained results for the relevant features, which are involved in the analysis criteria of this thesis work. In the table, letters Y and X mean yes and no supported, respectively for the results. The relevant features are shown in columns as follows:

- Regarding the features of the challenge of involving users to develop or modify their own applications (EUD): User Participation (UP), System-Aided (SA), and Technique (T). In the T column, letters represent the first letter of the EUD technique or metaphor that has been applied: Programming by Example (PbE), Magnetic Poetry Metaphor (MPM), Jigsaw Metaphor (JM), Butler Metaphor (BM) and Natural Programming (NP).

- Regarding the features of the challenge of involving users in modeling tasks: Model-Based (MB), Profile who performs modeling tasks (P). In the P column, letters ME, U and B mean Modeler Expert, Users and Both, respectively.

- Regarding the features of the challenge of achieving non-intrusive interoperability between models of heterogeneous modeling approaches: Different modeling Approaches (DA), Interoperability mechanisms (I), Intrusive with Meta-Models (I-MM), Collaborative Modeling mechanisms (CM), Tool Support (TS).

- Regarding the feature of the Application Domain (AD), it points out whether the approach has to be applied in some Specific domain (S) or it is General (G), which means that it is domain-independent.

- Regarding the feature of the Application Process (AP), it indicates if a process is provided for the application of the approach.

	EUD			Modeling		Interoperability						
	UP	SA	T	MB	P	DA	I	I-MM	CM	TS	AD	AP
PiP	Y	Y	PbE	X	X	X	X	X	X	X	S	X
a Cappella	Y	Y	PbE	X	X	X	X	X	X	X	G	X
CAMP	Y	Y	MPM	X	X	X	X	X	X	X	G	X
The Accord toolkit	Y	Y	JM	X	X	X	X	X	X	X	S	X
Alfred	Y	Y	BM PBE	X	X	X	X	X	X	X	G	X
The spreadsheet paradigm	Y	Y	NP	X	X	X	X	X	X	X	G	X
The Whyline	Y	Y	NP	X	X	X	X	X	X	X	G	X
PANTO	Y	X	NP	Y	ME	X	X	X	X	X	S	X
BaVeL	Y	Y	X	Y	U	X	Y	Y	X	Y	G	Y
Voelter and Solomatov	X	X	X	Y	ME	Y	X	Y	X	Y	G	Y
Giachetti	X	X	X	Y	ME	Y	Y	Y	X	Y	G	Y
Guerra et al.	X	X	X	Y	ME	Y	Y	X	X	Y	G	Y
Klar et al.	X	X	X	Y	ME	Y	Y	X	X	Y	S	Y
Kappel et al.	X	X	X	Y	ME	Y	Y	X	X	Y	G	Y
Invar	Y	X	NP	Y	B	Y	Y	X	X	Y	S	Y

Table 3.17: Summary of the state of the art by showing the analyzed features

As the table shows, most of the analyzed approaches use Programming by Example or Natural Programming as EUD technique to involve users in the creation or modification of a software artifact. On the one hand, the Programming by Example technique requires that users show the desired behavior to the system and select the relevant events, which can be tedious. In addition, users cannot physically do all they may want to be automated. Therefore, the Programming by Example technique is not always appropriate. On the other hand, the Natural Programming technique and providing closer languages seem to be the most extended option to involve users in the customization of a software artifact.

Overall, EUD techniques provide better user participation and involvement than the rest of approaches. However, most of these approaches limit

the expressivity and capacities to users. For this reason, most of the studied EUD approaches are only appropriate for developing simple tasks commonly described in the literature, such as controlling lights. Other approaches lack control structures such as conditionals, which only allow users to program basic event action rules.

In MDD, a multitude of modeling tools is available supporting different tasks, such as model creation, model transformation, and code generation. However, it is often difficult to use tools in combination or involve users who have different background in the descriptions of models due to both the lack of interoperability mechanisms and the necessary modeling skills that users may have. Therefore, the potential of MDD could not be fully exploited.

As Table 3.17 shows, none of the studied approaches attempt to confront the relevant features of modeling and interoperability. First, some of these approaches provide interoperability mechanisms but they are intrusive with the structure of meta-models. Second, some of these approaches are designed to be applied in specific domains such as variability models, so their application cannot be transferred to existing modeling approaches of different domains. Third, most of these approaches only involve Modeler Experts in modeling tasks. Although only one of the approaches (the Invar approach) involve both modeler experts and users in the description of the system in models, it limits the participation of users by only enabling them to select features from a predefined set, which is previously designed by modeler experts.

Unlike these approaches, this thesis work claims for an approach in which different users actively collaborate for obtaining a unified system description in models using different modeling approaches. The collaboration of different types of users such as end-users is really important to minimize the mismatch between their expectations and the system behavior. Moreover, collaboration

favors the use and adoption of the system. Note that none of the studied approaches provide collaborative modeling mechanisms (see the CM column in Table 3.17) since these approaches are focused on transformations from an entire model to another model. Moreover, these approaches do not provide collaborative modeling mechanisms to determine in which model concerns a different user may be involved. Therefore, these collaborative mechanisms become critical to obtain a full system description in models that combines model descriptions from different users and modeling approaches.

Recently, the Collaboro [116] approach provides a collaborative environment that enables the discussion among developers and users to define a DSL. In Collaboro, developers and users have the chance to request changes, propose solutions and give an opinion (and vote), which will be accepted/rejected whether an agreement is achieved. This discussion enriches the definition of both the abstract (i.e., meta-model) and concrete (i.e., notation) syntax of a DSL until no more changes are requested. At the end, a DSL definition is obtained that ensures that the end result satisfies as much as possible the expectations of the end-users.

In addition, an international effort known as the GEMOC Initiative [117] has emerged to explore the development of techniques, frameworks, and environments in order to facilitate the creation, integration, and automated processing of heterogeneous modeling languages. The GEMOC Initiative highlights the importance of supporting users with different levels of experience to express their perspective using their own language. This initiative also stresses the importance of integrating heterogeneous parts to deliver a global service. Thus, the GEMOC Initiative reinforces the contribution that this thesis work claims since, to the best of our knowledge, there is no approach that simultaneously does the following: 1) enable collaborative modeling to delimit the aspects of the system that may be described by another user

who describes those aspects with models using a different modeling language; and 2) integrate model fragments that have been described using a different modeling language in a non-intrusive way (i.e., without affecting the structure of modeling languages) to obtain a model with a unified system description.

Therefore, in spite of the research efforts that have been done, this chapter shows that there is still work to be done in order to completely solve the challenges confronted in this thesis.

Chapter 4

ADDRESSING THE INVOLVEMENT OF USERS

Nowadays, a select few have access to be actively involved in MDD processes since users face barriers and challenges (e.g., steep learning curves, arduous concepts and user interfaces) that make the description of domain-specific content hard for them. In addition, it becomes necessary to provide a collaboration to share knowledge in MDD processes in order to develop a new generation of software systems that requires expertise in a variety of domains. Among other benefits, the collaboration promotes a continual validation of the software to be built [118], thus guaranteeing that the final software will satisfy the users' needs [116]. Therefore, it is very important that different types of users participate cooperatively from the very beginning [11, 12]. Although several approaches have confronted collaborative modeling, most of them address collaboration to involve different roles using the same modeling language and tools. This work goes one step further by achieving collaborative modeling since it involves different types of users in modeling tasks using different modeling languages for describing system properties that depend on them.

The remainder of this chapter is structured as follows: Section 4.1 identifies both the most appropriate phase of the MDD process to actively involve users, and the main issues that need to be addressed in that phase. Section 4.2 presents collaborative modeling, its challenges, lessons learned, and design decisions. Section 4.3 overviews our proposal and its main building blocks to involve users in modeling tasks in a collaborative way. Section 4.4 shows how our proposal has been put into practice and validated throughout several case studies. Finally, Section 4.5 concludes the chapter.

4.1 Identifying the phases of MDD processes and issues

To address the involvement of different types of users, we first identify the most appropriate phase of the MDD process to actively involve them. Afterwards, we identify the main issues need to be dealt with.

Figure 5.3 shows a highly simplified view of the different phases of a MDD process. First, the requirements of the system are represented using Platform-Independent Models (PIMs). Then these PIMs are converted into Platform-Specific Models (PSMs), which, in turn, are converted into code.

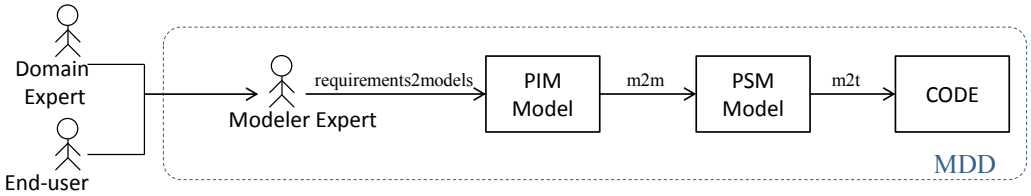


Figure 4.1: Classic approach of a MDD process

PIMs are used to model the functionality and structure of the system independently of the technological details of the platform upon which it will

be implemented, while PSMs are used to combine the specifications contained in the PIM models with the details of the platform that is chosen to implement the system [40].

Therefore, the platform-independence of PIMs makes them the most appropriate to address the issues identified above to actively involve users in MDD processes.

To support the participation of users in the description of PIMs, we identify two main issues: scoping the user-dependent participation and specifying user-dependent properties.

- **Scoping the user-dependent participation.** Since many of the users involved in modeling tasks are not familiar with describing the full aspects of the system [13] (e.g., end-users cannot pay attention to describe software quality aspects the way as software engineers do), it is necessary to identify and delimit which aspects of models may be described by users. Therefore, in order to obtain a unified system description, users should be provided with collaborative modeling mechanisms that allow them to focus on modeling their dependent properties rather than on modeling the functionality and structure of the entire system.

- **Specifying user-dependent properties.** Traditional interviews are still used to capture the user needs in software development activities [119]. However, traditional interviews are not always the best option for extracting user needs [120].

  Works like [121] show that there are still problems in the extraction of users' needs. One of the most important problems is the one related to **problems of understanding**. These problems result from the necessary involvement of different types of users such as requirements

analysts, designers, developers, and end-users. The requirements are produced and interpreted by people with different experience levels and backgrounds. For instance, end-users do not understand the jargon of software developers and developers often do not understand the jargon of end-users [13]. This makes the adoption of models hard for users since they face barriers and challenges [1] (e.g., steep learning curves, arduous concepts and user interfaces) in order to participate in software projects of MDD processes.

Therefore, mechanisms need to be provided to scope the user-dependent participation and to overcome the problems of understanding that users face to specify their user-dependent properties.

4.2 Collaborative Modeling

In order to address the issue of scoping the user-dependent participation that was identified in the previous section, we have studied collaborative modeling. The growing number of modeling approaches underlines the rising relevance of developing and introducing collaborative modeling mechanisms that give different users the opportunity to contribute in model descriptions.

Collaborative modeling is defined as follows[122]:

Collaborative modeling refers to a process where a number of people actively contribute to the creation of a model.

Collaborative modeling has been a research topic since the late 70's and becomes important with 1) increasing need for collaboration among modelers and domain experts [123], and 2) increasing complexity of systems and organizations [6]. Since the late 70's, various other modeling approaches have adopted the notion of collaborative modeling such as [16, 116].

Nevertheless, the current state of practice of collaborative modeling approaches confronts collaborative modeling mechanisms using the same modeling language and tools among all the users who actively participate in the modeling effort. Figure 4.2 depicts our approach compared with the current state of practice.

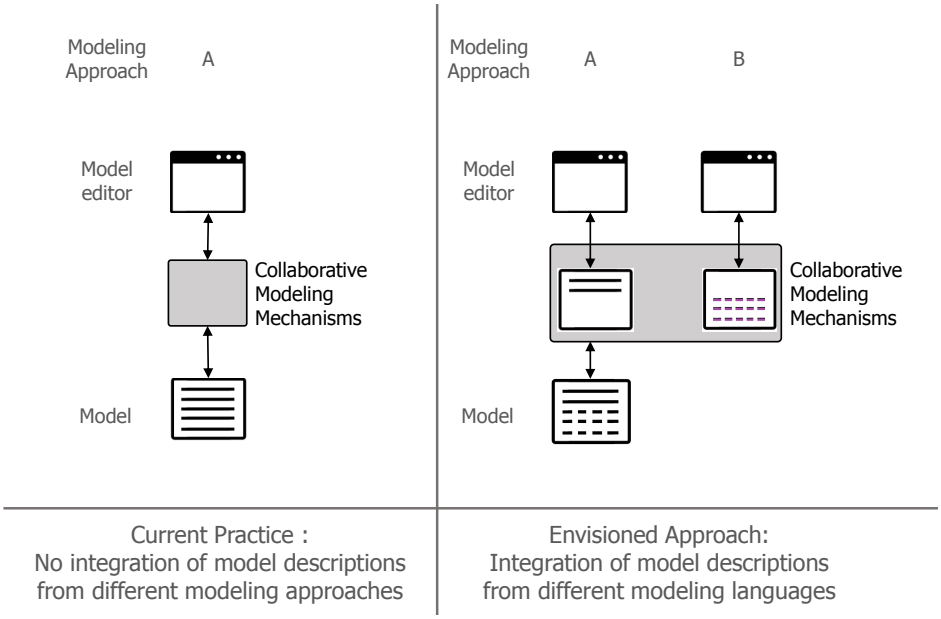


Figure 4.2: A highly simplified view of collaborative modeling: current state of practice (left) and our approach (right)

In the current state of practice (see left side of Figure 4.2), traditional collaborative modeling approaches have used a homogeneous software process and toolset. Moreover, they usually enable regular and proactive face-to-face meetings, and team members usually have the same language and work culture [3, 124]. Specialists within teams need to exchange knowledge among themselves and across team boundaries. Traditional software tools usually provided limited collaboration support features such as shared workspaces and file repositories [3].

In our envisioned approach (see right side of Figure 4.2), collaborative

modeling mechanisms seeks to address the issues identified in Section 4.1 (scoping the user-dependent participation and specifying user-dependent properties) from different modeling languages. In particular, collaborative modeling mechanisms seek to support in our approach that some concerns of the system are described in models using a modeling approach and model editor (see the column A in the right side of Figure 4.2) and other concerns are described using a different modeling approach and model editor (see the column B in the right side of Figure 4.2) that fits users' context and needs in order to overcome the barrier of users to describe domain-specific content in models. Therefore, our approach 1) aims the involvement of users to describe themselves their user-dependent properties, and 2) obtains a unified model that integrates descriptions of both modeling languages.

For supporting the above mentioned collaborative modeling mechanisms in our approach, it is important to identify the **key and critical challenges** in the collaborative modeling field to overcome them. These challenges are the following [6]:

- The integration of submodels or models descriptions that are made from different participants who actively participate in the modeling effort, and the resolution of conflicts during the integration of such submodels or models descriptions.

- The lack of modeling skills avoids that participants are actively involved in the modeling effort.

- The design of an approach for supporting the collaborative modeling effort (i.e. a sequence of steps) that can be applied in existing modeling approaches.

In addition, it is important to identify **the lessons learned** in the collaborative modeling field to take them into account in our approach. The

lessons learned are the following:

- The involvement of users with guidance throughout the process becomes important since users' participation certainly helps to relate the models with the real needs [5].
- Users can also fulfill different roles in the collaborative modeling process to coordinate tasks whereas in traditional modeling methods, the input of users is processed into a model by the analyst/modeler [6].
- The process is initiated by managers to drive the process in the beginning, and the first challenge usually is engage the right stakeholders in the process [5].
- The use of a preliminary model is extended in the so-called prototyping strategy, where for each step in the modeling process an analyst prepares the model and participants subsequently criticize and change themselves the model [125].
- The modeling process should be iterative since models should be improved during the modeling process [5].

The challenges and lessons learned presented above inspire **design decisions** of our proposal for supporting collaborative modeling as follows:

- Variability management will be used in a novel way for enabling collaborative modeling. Thus, our approach will enable a user, who acts as a manager, to: 1) prepare a model with the commonalities of the system, and 2) determine variabilities in which a different user is engaged in the modeling effort. Moreover, the application of variability management in our approach provides guidance throughout the modeling process to the users, who are engaged in the modeling effort, with information

about the system concerns they may describe. Thus, our approach will provide key operations and queries on variability models to manage and integrate model descriptions from different users, and detect conflicts during the integration of such model descriptions.

- Users will collaborate themselves in the modeling effort using a language which is familiar for them. This mitigates the lack of modeling skills that users face to actively participate in the modeling effort, and different roles are supported in the modeling process. Moreover, this design decision is inspired by the guidelines to involve users in modeling tasks that are presented in Chapter 5, which states that users should be provided with a closer language and tools.

- A modeler expert will initialize and execute our approach in order to 1) specify the correspondences among concepts of the different modeling languages, and 2) provide the proper tool support for users, who may be involved in the description of user-dependent properties.

- The modeling process will be iterative. Thus, our approach supports during the modeling process the modification of the initial model, the creation of new variabilities, and the modification/description of variabilities.

Next, we define our collaborative modeling process that is comprised of the following five stages: *Identify project goals, Identify users, Choose modeling approaches, Specify and execute our proposal*, and *Build models in a collaborative way.* Figure 4.3 shows each stage as a rounded square. The first two stages are carried out by the *Modeler expert* using interviews for identifying project goals and users is described in Section 5.1.

In the third stage, the *Modeler expert* selects the two appropriate modeling approaches for enabling collaborative modeling and interoperability

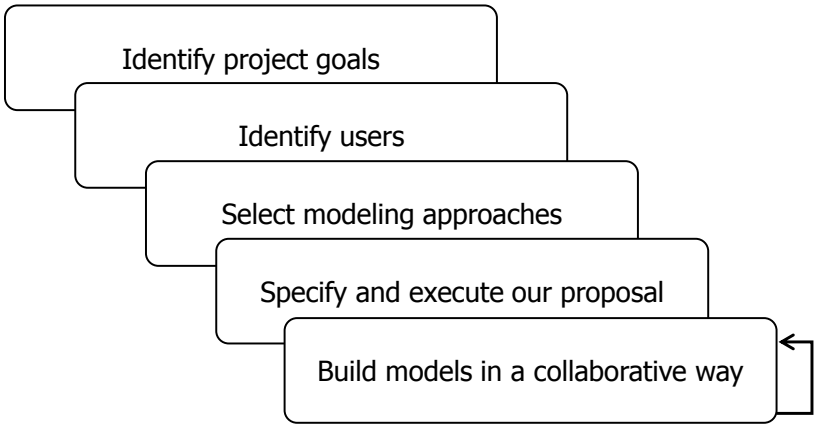


Figure 4.3: Different stages of the collaborative modeling process

in modeling tasks. Selecting the appropriate modeling tool is one of the most important phases of any modeling exercise [5]. Model selection should be driven by the goals of project and the users. It is important that the choice of modeling tools should happen with the users after the goals are decided and after surveying the available tools and selecting the ones that are most appropriate. In fact, the *Modeler expert* may select the modeling approaches that users are most familiar and comfortable with [5].

At this point, it is also worth pointing out that although this work promotes that the *Modeler expert* may select existing modeling approaches among the broad variety of modeling approaches, it is not always available an existing modeling approach that both fits the goals and is closer to the users of a concrete project. In this case, the *Modeler expert* should design a language for actively involving users in the modeling effort. To do this, the *Modeler expert* may follow the guidelines to involve users in modeling tasks that are identified in Section 5.2.

In the fourth stage, the *Modeler expert* will initialize and execute our approach by taking as input the two selected modeling approaches. Variability management provides our approach with mechanisms for supporting

collaborative modeling by scoping the user-dependent participation, whereas the interoperability mechanisms provide our approach elements that overcome the barriers of users to specify themselves the user-dependent properties using one of the selected modeling approaches.

In the last stage, the users build models in a collaborative way using the two selected modeling approaches (e.g., the modeling approach A and the modeling approach B as was depicted in Figure 4.2). On the one hand, the process is started by a user who acts as a manager and drives the process since s/he prepares the model (as we have identified in the lessons learned in collaborative modeling) using the model editor of the modeling approach A and sets those model elements in which a different user should be actively engaged to describe them (user-dependent properties). On the other hand, the different user describes the properties using the model editor of the modeling approach B. Our approach aims to provide operations and queries that automatically integrates the model descriptions of the modeling approach B into the modeling approach A. As a result, collaborative modeling from two different modeling approaches is supported. This stage is iterative until modeling tasks (see the recursive arrow of Figure 4.3) are finished.

The next section overviews how this thesis work addresses the two last stages of the collaborative modeling process (*Specify and execute our proposal*, and *Build models in a collaborative way*).

4.3 Overview of this work

The necessity of involving users in modeling tasks and integrating their model descriptions becomes crucial. In this thesis, we deal with this necessity by not only enabling users to describe themselves their user-dependent properties in a non-intrusive way (i.e., without affecting the structure of modeling languages)

but also, encompassing collaborative modeling throughout the modeling effort by scoping the user-dependent participation.

To start with, we propose an approach that addresses the issues identified above (scoping the user-dependent participation and specifying user-dependent properties) to involve end-users in modeling tasks. Specifically, our approach provides end-users with a tool-supported visual modeling language that enables collaborative modeling using the variability management facet of feature models. With the tool, end-users are able to select and customize themselves system features using concepts that fit their skills, context and needs, which helps them to overcome their understanding barrier with a MDD process for developing pervasive systems, which has been used to apply the approach.

To achieve this, we start with the identification of user skills and their software activities as well as guidelines to involve users in modeling tasks. In short, the main conclusions of this identification are that users do not have to be transformed into modeler experts, and users should be provided with closer languages since in far too many cases are inclined to favor the tools that users are most familiar and comfortable with [5, 7]. In addition, the EUD techniques and metaphors that were presented in Chapter 2 and 3 serve to identify interface design decisions. These interface design decisions can be applied to create a modeling environment that fits users' goals and needs since closer languages are not always available for involving users in modeling tasks within the existing variety of modeling approaches.

However, this approach of selecting and customizing system features presents some drawbacks and it is not enough to involve different types of users in existing MDD processes, who could require more expressiveness to describe their user-dependent properties. To address this, we propose an approach that uses the variability management facet that uses models to

describe variation points rather than features. Thus, we enable collaborative modeling by supporting both the selection of model fragments of the system that may be described using a different modeling language, and the integration of those model fragments once they are described.

Figure 5.3 shows an overview of our envisioned approach to involve users in description of PIMs of MDD processes. First, a user (e.g., a Modeler expert) describes requirements in a base PIM model of a MDD process. Second, this user scopes the user-dependent participation by defining a set of gaps in models using variability mechanisms. Finally, a different user (e.g., an end-user) specifies the user-dependent properties using PIM model fragments of a different modeling language. At the end of this process, a PIM model that unifies model fragment descriptions is obtained even when these fragments have been described using a different modeling language.

Therefore, different users do not have to deal with unfamiliar concepts, and they are provided with mechanisms that let them know which aspects of the system they are involved in. This can eliminate some of the barriers that exist in the description of domain-specific content, which could help to achieve a wider adoption of MDD processes in industry [1, 4].

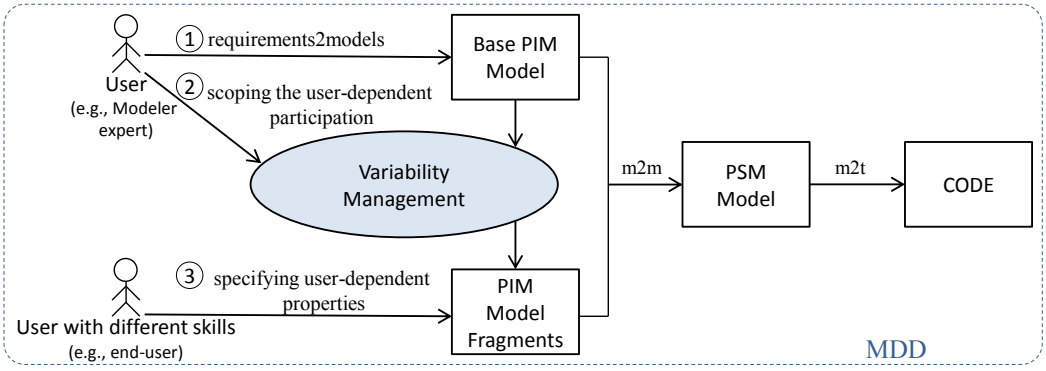


Figure 4.4: Overview of our proposal

Afterwards, we provide mechanisms that allow models from different modeling approaches to interoperate in a non-intrusive and collaborative way. As explained in Chapter 3, although several approaches have dealt with the pursued goals, they still present some drawbacks. To overcome them, we propose a method, named Medem, which enables collaborative modeling by bridging two different modeling approaches. Specifically, Medem enables the user of an existing modeling approach to define a set of gaps. Another user can fulfill these gaps using models of a different modeling approach. To do this, we combine interoperability and variability mechanisms in a non-intrusive way for the existing meta-models. On the one hand, we apply interoperability mechanisms by means of (1) a weaving model that links model concepts of each approach, (2) model transformations that obtain model descriptions from one model to another, and (3) model queries that manage information of models. On the other hand, we apply variability management mechanisms in a novel way to determine gaps that may be fulfilled by the new user.

From the two facets to model variability (features and models for managing the variability of products) that were presented in Chapter 2, we choose the facet of models for managing the variability of products using a separate variability language rather than annotating the base modeling language. Thus, the structure of the modeling language does not require to be extended (non-intrusive) and variation points can be viewed as gaps that have to be described using model fragments of a different modeling language.

In order to turn into reality the proposal, a toolkit was developed. The toolkit enables the symbiosis between interoperability mechanisms and variability modeling. Specifically, the toolkit implements operations that are in charge of managing the creation and description of gaps, which also imply other operations such as queries and transformations of model fragments from a modeling approach to the another one. These operations are implemented

using the widespread tools of the Eclipse Modeling Project[1] in order to promote the application of our approach in existing modeling approaches. Although these operations are implemented to be domain-independent, the toolkit has to be initialized by a modeler expert in order to provide the toolkit with domain-dependent information such as the weaving model.

For validation purposes, the proposal has been applied in three case studies from different domains and levels of complexity. In particular, the case studies involve users in modeling tasks in a non-intrusive way of existing modeling languages within the following domains: smart home systems, web information systems, and biomechanical protocols.

Figure 4.5 presents the main building blocks that support the proposed approach. Each building block is denoted by a rounded rectangle within the Medem block, which represents our approach. As figure shows, each building block is related to the fields that were explained in Chapter 2 (Model-Driven Development and Variability Management), which is colored gray at the top of the figure. In addition, the left side of the figure shows the two existing modeling languages that are necessary to apply our approach. By *Modeling language*$_a$, we refer to the modeling approach which integrates of model descriptions of another modeling approach that we identify as *Modeling language*$_b$. The main building blocks of the approach are: interoperability mechanisms and Collaborative modeling.

- **Interoperability mechanisms.** This building block is related to the Model-Driven Development field, which enables us to exchange model descriptions between the *Modeling approach*$_a$ and the *Modeling approach*$_b$ by means of model transformations. In particular, we use the hybrid approach for model transformations that was described in

[1] `http://www.eclipse.org/modeling/`

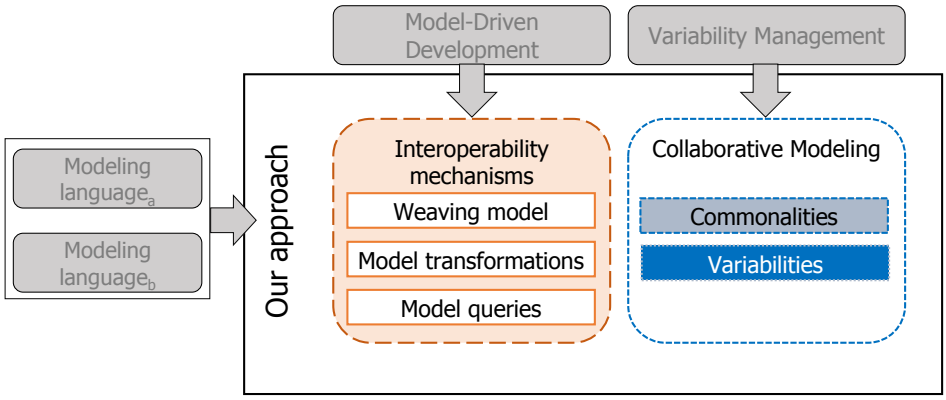


Figure 4.5: Main building blocks of our approach

Chapter 2 in which a weaving model is designed to set correspondences among the concepts of the modeling approaches, and transformation rules are created according to the weaving model for enabling the automatic translation from one approach to another. Our decision to use this hybrid approach for model transformations with a weaving model and transformation rules comes for the following reasons:

- It is the most followed by the most adopted languages [22].
- It becomes popular and useful tools in research and industry [71].
- It is non-intrusive with the structure of models of the modeling approaches.

- **Collaborative modeling.** This building block is related to the Variability Management field, which enables us to manage a range of products by specifying variable elements on model descriptions. Usually in the variability management field, *commonalities* refer the product elements that come up across all feasible product configurations meanwhile, *variabilities* refer the product elements that can be replaced

(gaps and gap descriptions as explained in Chapter 2). However, *commonalities* and *variabilities* use to be described using the same modeling approach and tools. In this work, we propose to use *variabilities* in a novel way to determine gaps that may be described using a different modeling approach. Thus, variability modeling mechanisms enable us to provide collaborative modeling.

4.4 Validation

The presented work has been validated to prove its applicability and feasibility throughout different domains and levels of complexity. In particular, three case studies have been developed following the guidelines for case study research by Runeson and Höst [126]. These case studies are introduced as follows:

1. ***PervML-Pantagruel.*** This case study tackles the application of Medem in an existing modeling approach for developing smart home systems in order to involve users in the modeling effort using another existing modeling language.

2. ***UIM-Sketcher.*** This case study addresses the application of Medem in two existing modeling approaches for two involving different roles of an organization in the development of web information systems.

3. ***Bioengineering kinematic - Medical Protocol.*** This case study involves doctors with biomedical engineers in the description and analysis of biomechanical protocols in existing tools. To achieve this, we address the design of a new Domain-Specific Language (since, to the best of our knowledge, there is no an existing DSL that fits the concepts of biomechanical protocols that doctors use) by following guidelines and

design principles from the EUD literature. Afterwards, this case study performs the application of Medem to involve doctors' descriptions within biomedical engineers' descriptions.

Overall, the evaluation of the case studies revealed positive results of our approach since it: (1) can be applied in different domains, (2) is non-intrusive with the existing the modeling languages, and (3) involves users in modeling tasks in a collaborative modeling way.

4.5 Conclusions

Achieving the involvement of users in MDD processes becomes necessary to enrich the description of models, and to help to adopt MDD processes by the software industry. To achieve this, non-intrusive interoperability between models of heterogeneous approaches becomes crucial in current industrial settings since different types of users, who have different background (such as engineers and end-users), should be involved in the modeling effort using a different modeling languages that fits their context and needs.

In this thesis the End-user Development, the Model-Driven Development and the Collaborative Management fields are combined in order to achieve 1) interoperability in a non-intrusive way, and 2) collaborative modeling from different modeling languages. Therefore, the identified issues (scoping the user-dependent participation and specifying user-dependent properties) can be tackled using variability management mechanisms in a novel way.

Chapter 5

INVOLVING END-USERS IN MODELING TASKS

Although the active involvement of different types of users in the description of PIMs of MDD processes is key as motivated in previous chapters, users use to transfer their requirements to a *Modeler expert* rather than participate themselves in modeling tasks. This is because models represent barriers to users since models can have concepts that are unfamiliar to some users, so users need the *Modeler expert* to describe the domain-specific content in models.

In order to allow users to describe themselves domain-specific content in models, it is necessary to identify the user skills and their software activities to set the target of our proposal. Next, it is necessary to identify general guidelines and interface design decisions from the EUD literature that make the participation of users easier in the description of system behavior in order to apply them for lowering barriers of users in the description of domain-specific content in models.

Afterwards, we apply the identified guidelines and design principles to provide a tool-supported visual modeling language that addresses the

involvement of end-users in modeling tasks of an existing MDD process for developing pervasive systems. We choose end-users and pervasive systems as application example because end-users are the ones who have more in-depth knowledge about both the services that must be provided by the system and the environment in which the system is going to be deployed. However, end-users face barriers to actively participate in the description of their system because they lack the skills to manage the technologies that the existing MDD process uses, so they have to transfer their requirements to a software professional.

The tool-supported visual modeling language not only enables the active participation of end-users in modeling tasks but also, enables collaborative modeling by addressing the issues that were identified in Chapter 4 (Scoping the user-dependent participation and Specifying user-dependent properties) using the variability management facet of feature models.

This chapter is structured as follows: Section 5.1 identifies user skills and their software activities. Section 5.2 identifies general guidelines and design principles to involve users in modeling tasks. Section 5.3 presents our tool-supported visual modeling language that applies the identified guidelines to enable end-users to collaborate in modeling tasks of an existing MDD process for developing pervasive systems. Finally, Section 5.4 presents the conclusions of the chapter.

5.1 Identification of User Skills and their Software Activities

Although the user population is quite diverse because they are present in a lot of domains and with different needs, users and the activities that they usually perform with computers have been analyzed in previous works [29] [127].

Two classes of user activities were identified whether they are involved in the creation or modification of a software artifact. More specifically, Class 1, which refers to modifying a software artifact, includes activities that allow users to choose among alternative behaviors (or presentations or interaction mechanisms) that are already available in the application by setting some parameters; these activities are usually called parametrization, customization, or personalization. Class 2, which refers to creating a software artifact, includes all the user activities that imply some programming in various programming paradigms.

These two classes can be supported by different types of interfaces: closed-option and open-option. Class 1 could be supported by closed-option interfaces and Class 2 could be supported by both interfaces. Closed-option interfaces provide users with a catalogue of requirements. This catalogue allows users to select those requirements that satisfy their needs. Open-option interfaces allow Class 2 to define new requirements if the requirements catalogue does not satisfy users' needs.

More recently, Fischer and Ye proposed a spectrum of software-related activities [127]; Figure 5.1 is adapted from that work and shows the spectrum of the above activities graphically. On the right side of the spectrum are the *Software Professionals*, i.e., software engineers that develop software systems for users other than themselves. On the opposite side (left side) are the *Pure End-users* that passively use software systems to accomplish their daily tasks. Users who are willing to perform various activities that cause them to modify and/or create software artifacts are represented in Figure 5.1 as *End-users who customize*; they may have certain software development skills, but they only develop software to solve the specific problems that they face.

Our target is to provide mechanisms for both *End-users who customize* (from now onward end-users) and *Software Professionals* (SPs) to allow them

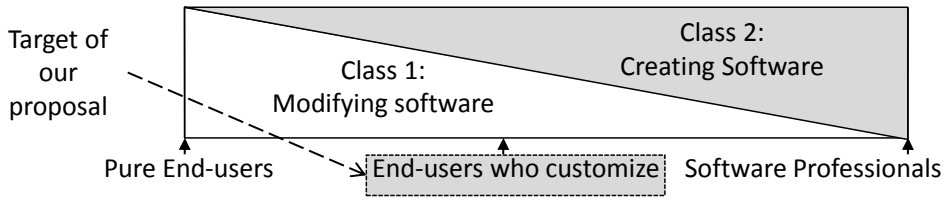


Figure 5.1: The spectrum of software-related activities

to work cooperatively in modeling tasks of MDD processes. Thus, the design of models is complemented by both (1) *Modeler experts*, who play the role of SPs' in MDD processes, contribute to improve the system behavior; and (2) end-users contribute by describing themselves their demands and desires, which helps end-users to successfully adopt and use the system [128].

5.2 Identification of Guidelines to Involve Users in Modeling Tasks

Once the target users are identified, we have studied the End-user Development literature to establish guidelines that make the participation of users in the description of system behavior easier. The guidelines are the following:

- **Users should be provided with a closer language such as a DSVL** to lower the barriers for users in the description of domain-specific content [7]. Furthermore, the use of a visual language seems to be the best option since visual languages have proven to be more intuitive and easier to use than other options like textual languages or general purpose languages. This is because the gap between the mental model of the user and the concepts of the DSVL is smaller than the concepts that SPs manage in a DSL [129, 130].

- **Users do not pay attention to software quality as software**

engineers do [13]. They do not bother at all about software engineering issues such as quality and maintenance. Therefore, users should focus on user-dependent properties, whereas software engineers should focus on engineering issues.

- **Users have to use a library of components as a starting point in order to customize their system**. In the context of End-user Development, it is essential that a library of components or a initial system be provided by SPs [131].

- **Users do not have to be transformed into SPs [8].** The plan is to provide techniques and tools that allow users to collaborate with SPs in the development of software systems.

- **Users have to be supported by specific tools made especially for them**. Nielsen [132] recommends that users should participate in the description of their system though user interfaces. These interfaces should "speak the user's language", and they should include good mappings between the user's conceptual model of the information and the computer's interface for it.

In order to provide users with a closer modeling language such as a DSVL, it becomes necessary to build an editor that enables the creation of models of that closer modeling language. To build the model editor, we have studied well-accepted techniques and metaphors in the field of End-User Development (such as Natural Programming and Visual Programming that were presented in Chapter 3). According to these studies, the main design interface decisions that may be applied in the design of a specific tool for users are the following:

- **Using a wizard:** the user needs to achieve a single goal (the description of their needed system) but several decisions need to be made before

the goal can be fully achieved (several steps), which may not be known to the user. Thus, the use of a wizard is recommended in [133] since the user wants to reach the overall goal but may not be familiar with or interested in the steps that need to be performed.

- **Offering navigation buttons:** navigation buttons suggest users that they are navigating a path with steps. This is recommended in [133] because the learning and memorization of the task of each step are improved. In addition, when users are forced to follow the order of tasks, they are less likely to miss important things and therefore will make fewer errors.

- **Displaying the elements using a grid layout:** this is recommended in [133] to any circumstance where several information objects are presented and arranged spatially within a limited area. This improves the presentation and it minimizes the time to scan, read and view objects on screen.

- **Offering options:** an interesting conclusion is reached in [134]: *what people see is what they select from!*. The study states that people tend to select from the entire list of options what they are first presented with. Rarely is an effort made to find additional options through scrolling. If eleven items are presented, the choice is from these eleven. When options must be compared among themselves, controls presenting all the options together will yield the best results.

- **Selection rather than introduce text:** the studies presented in [135] show the advantages and disadvantages of using either entry fields or selection fields for data collection. Since information became less familiar or subject to spelling or typing errors they recommend choosing a selection technique.

- **Using autocompletion:** The study showed in [135] states that aided entry, also known as autocompletion, is preferred over unaided entry methods, and it is also the fastest method. Autocompletion reduces errors in comparison to unaided entry. In addition, it also minimizes the user's effort by reducing input time and keystrokes.

- **Using a warning:** this is recommended in situations where the user performs an action that may unintentionally lead to a problem [133] and the system cannot or should not automatically resolve this situation so the user needs to be consulted. The warning might also include a more detailed description of the situation to help the user make the appropriate decision by means of two options at least.

- **Offering all options:** this is recommended when the number of options is not large and they can be displayed without scrolling [135].

- **Offering some options:** this is recommended when the number of options is high and it needs a scroll to be displayed. Thus, it is recommended to show some options of the available list [135]. This improves the speed of performance and satisfaction.

The *Modeler expert* can apply the identified guidelines and interface design decisions in order to involve users in modeling tasks by providing them with closer modeling languages and closer model editors in case that there is not available an existing modeling language and/or model editor that fit both the project goals and the users to be involved in modeling tasks.

5.3 Applying the identified guidelines and interface design decisions to pervasive systems

In order to apply the guidelines and interface design decisions identified above, we use the pervasive systems domain. Since end-users could not participate in the description of their pervasive system (e.g., an smart home) in an existing MDD approach because they lack the skills to use the technologies to describe in models the services that must be provided, a closer modeling language and tools must be provided.

Moreover, the issues that have been identified in Chapter 4 (Scoping the user-dependent participation and Specifying user-dependent properties) may be addressed. To achieve this, we propose to combine MDD and Software Product Lines (SPLs). Specifically, we propose the use of the variability management facet of feature models.

In SPLs, many efforts have already been made to improve the development of a large range of software-intensive systems faster, better, and cheaper [81] in different domains such as smart home systems [136, 137].

There are several works that show how to combine MDD and SPLs [138, 89]. For example, Voelter and Groher [89] describe an approach where development is combined with model-driven development. They define aspects at the modeling level, the transformation level, and the implementation level. They apply their approach to the Smart Home domain. Anastasopoulos et al. [138] apply a combination of both MDD and SPL to the Ambient Assisted Living (AAL) domain. They express variations in smart home functionality as features, and synthesize AAL specifications by composing features. Compared to our work, the above approaches do not

actively involve end-users in the modeling effort of the MDD-SPL system, which is essential for the successful development in many domains such as smart homes [139].

In our initial approach, we actively involve end-users in modeling tasks by providing both: 1) a feature model that addresses the identified issue of scoping the user-dependent participation by both determining the variabilities of the system and guiding the users' collaboration throughout modeling tasks, and 2) a closer language for end-users that fits end-users context and needs by applying the identified interface design decisions in order to address the identified issue of specifying user-dependent properties.

Considering the schema of the MDD-SPL that Figure 5.2 shows, a product operation transforms input assets into an output system according to the configuration specified in a decision model. This approach contributes with an end-user tool that uses a closer language and enables users to themselves collaborate in the modeling effort by configuring variabilities of the system. The variabilities are stored in the decision model, which drives the production process. The design of the commonalities of the system and the variabilities (the feature model) is performed by the *Modeler expert* (who plays the role of *Software Professional*). Our tool provides model queries and transformations to obtain the output system.

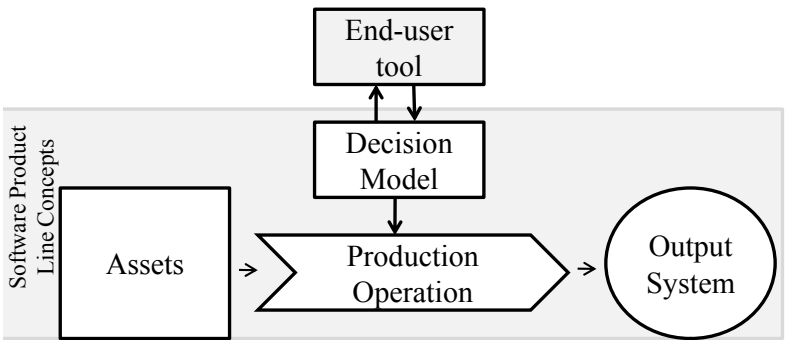


Figure 5.2: The initial approach for involving end-users in modeling tasks

This approach has been applied in an existing MDD approach for developing smart home systems [140, 141]. Specifically, we have extended a SPL to allow end-users to create tailored solutions that directly reflect their needs and expectations using a closer language. Figure 5.3 illustrates the blocks used in this application. The input assets consist of a collection of models describing all smart homes that can be produced. These models are created by using the PervML language [99]. A smart home is uniquely defined by the selections made on the feature model, which plays the role of decision model. The selected features determine which elements of the PervML models are used for the initial configuration of the smart home by means of a Realization Model. Finally, the output system is obtained through a model transformation.

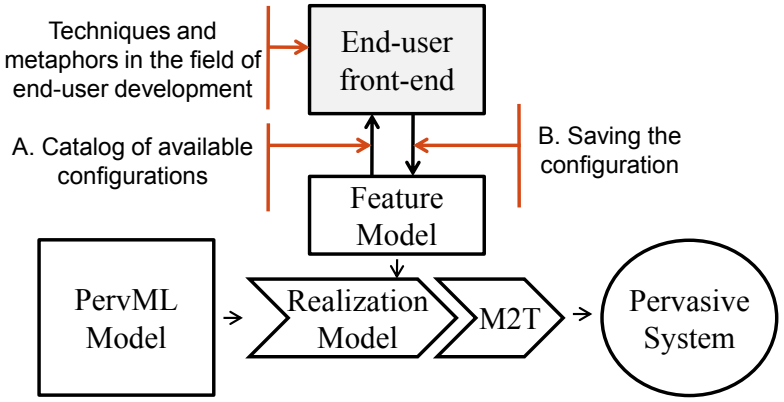


Figure 5.3: Application of the approach using features

The *Software Professional* sets the smart home commonalities and variabilities by means of the feature model making assumptions about the desirable functionality of end-users. Conversely, end-users are the ones who best know their activities and their functionality expectations. As described in the identification of user skills in Section 5.1, end-users and *Software Professionals* actually possess distinct types of knowledge. End-users are the "owners" of the problem and experts are the "owners" of the technology

to solve the problem. Although end-users are not experts, they have deep knowledge of their specific environment and they should be able to develop their own smart home system according to their needs. Hence, we involve end-users in the Smart Home configuration in order to minimize the mismatch between user expectations and system behavior.

In order to tackle this, end-users must be supplied with a closer language that allows them to describe their needs [14]. To the best of our knowledge there is no available an existing modeling language that fits with this application, so we have applied the identified guidelines and interface design decisions identified in Section 5.2 to develop a tool that allows end-users to configure their smart home system by themselves (see *End-user front-end* in Figure 5.3). In particular, the end-user tool allows end-users configure the feature model by selecting from a catalog of available options (see Step A in Figure 5.3) which services and devices must be available in each location. Thus, once end-users have finished the configuration, the operations provided in our approach will obtain the realization model (see Step B in Figure 5.3) that determines the output system by applying model transformations.

Next, details about the blocks involved in our approach are described as follows:

The Pervasive Modeling Language (PevML) model. PervML [142] is a DSL for describing pervasive systems using high-level abstraction concepts. This language focuses on specifying heterogeneous services in specific physical environments such as the services of a smart home. These services can be combined to offer more complex functionality by means of interactions. These services can also start the interaction as a reaction to changes in the environment. The main concepts of PervML are: (1) a *Service* coordinates the interaction between suppliers to accomplish specific tasks (these suppliers can be hardware o software

systems); (2) a *Binding provider* (BP) is a supplier adapter that embeds the issues of dealing with heterogeneous technologies; (3) an *Interaction* is a description of a set of ordered invocations between Services; and (4) a *Trigger* is an ECA rule (Event Condition Action) that describes how a Service reacts to changes in its environment. This DSL has been applied to develop solutions in the smart home domain [143]. The reader is referred to [99] for a detailed description of PervML concepts.

For example, the bottom of Figure 5.4 shows an abstraction of a PervML model that describes the blocks for the assembly of a smart home system [142]. The grey blocks implement the functionality of the selected features. The white blocks enable an alternative functionality of the system. The (l), (o), (m) and (p) blocks provide adapters for the new resources available.

The feature model. As described in Section 2.3, feature models are widely used to describe the set of products in a software product line in terms of features, which are hierarchically linked in a tree-like structure and are optionally connected by cross-tree constraints. There are many proposals for the type of the relationships and the graphical representation of feature models [144]. We have chosen the Feature Model [145] as the modeling language because it is feature reasoning oriented and has a good tool support [146].

For example, the top of Figure 5.4 shows a feature model that determines the initial and the potential features of the smart home. The grey features are selected to specify a member of the smart home family. The white features represent potential variants. Initially, the smart home provides *Automated illumination, Presence simulation and a Security* system. This security system relies on *In home* detection

(inside the home) and a siren alarm. The system can potentially be upgraded with volumetric presence detection and more alarms to enhance home security.

The feature model also determines how the features relate to each other by cross-tree constraints. As the feature model of Figure 5.4 shows, these relationships are: *Optional* represented with a small white circle on top of the feature, *Mandatory* represented with a small black circle on top of the feature, *Multiple choice* represented with a black triangle, *Single choice* represented with a white triangle, *Requires* which it is represented with a dashed arrow and *Excludes* represented with a dashed double-headed arrow.

The End-user front-end. It allows end-users to define the initial configuration of the smart home system by means of a closer language. To design it, we have chosen the *jigsaw metaphor* [12]. As described in Chapter 3, the "jigsaw pieces" metaphor is based on the familiarity evoked by the notion and the intuitive suggestion of assembly by connecting pieces together. Essentially, it allows users to take variability decisions through a series of left-to-right couplings of pieces. Constraining connections in a left to right fashion also provides users with the sense of a pipeline of information flow. Moreover, this end-user front-end applies all the guidelines that were identified in Section 5.2 and applies the following design interface decisions: displaying the elements using a grid layout, offering options, selection rather than introduce text, and offering all options.

The manner in which the end-user front-end is used in our approach is as follows: end-users define their initial configuration by means of the end-user front-end. The end-user front-end should present the whole

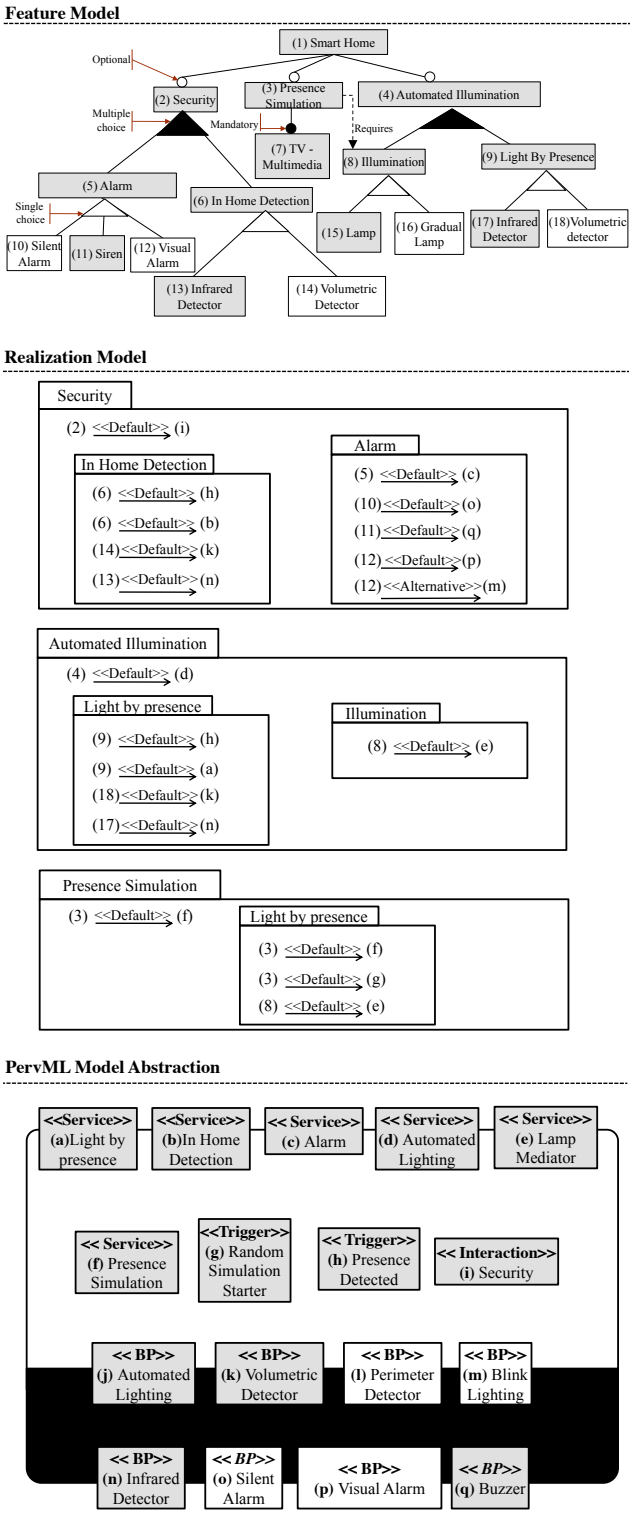


Figure 5.4: Models for the SPL

set of features defined in the feature model as Jigsaw pieces. The jigsaw editor should be divided in two areas. An area should contain compatible and non-compatible pieces and the other area should be the workspace where end-users can define their initial configuration. In the Figure 5.5, we illustrate each jigsaw piece which represents a feature of the feature model previously modeled in Step A by the *Software Professional* (see top of Figure 5.6). The root piece is filled in black with a gray frame, the compatible pieces are filled in black and the non-compatible pieces are filled in gray. When a jigsaw piece is added within the workspace, non-compatible pieces should be disabled and shadowed, indicating which pieces are compatible. Compatible and non-compatible pieces are defined by the feature model. Therefore, end-users should do the following steps to define an initial configuration:

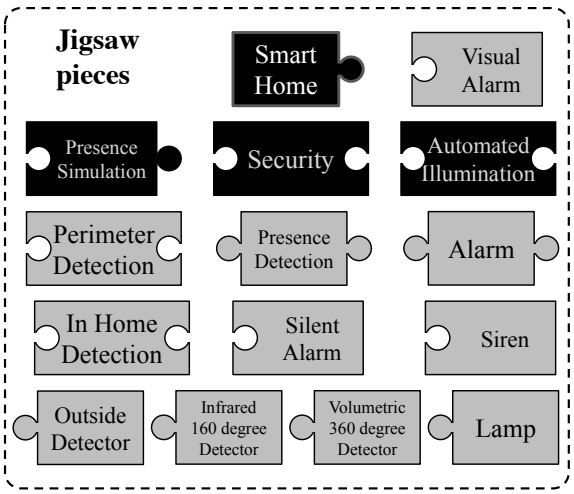


Figure 5.5: Defining the initial configuration

1. Select the root piece. From this feature end-users can define all their initial configuration services.

2. Add available pieces to the last piece selected. If end-users select a leaf piece a service will be configured.

3. Repeat steps 1 and 2 until all pieces have been selected or repeat until all the services needed are configured.

When the services have been configured, end-users will have a line of puzzle for each service initiated in the system from the root to the leaves. The services which are not initialized will not be available in the system.

According to the feature model and the initial configuration represented in Figure 5.6, end-users have defined three initial services: *Presence Detection, Alarm* and *Automated Illumination*. In the end, end-users will attain a line of puzzle for each service.

Realization model. It is an extension that we have incorporated to Atlas Model Weaving (AMW) [147] in order to relate the features to the PervML elements. AMW is a model for establishing relationships between models as was described in Chapter 2. Our extension augments the *AMW relationship* with the *default* and *alternative* tags. This augmented relationship is applied between features and PervML elements (BPs and Services). In the context of a BP, the *default* relationship means that the BP is selected for the initial configuration of the system. The *alternative* relationship means that the BP is considered a quiescent element that should be incorporated to the SPL product, but does not participate in the initial configuration.

This model (see the middle of Figure 5.4) establishes the relationships between the features and the PervML elements. For instance, the visual alarm feature is related to a BP (p) for visual alarms, but, alternatively, it can be replaced with a BP (m) that emulates the visual alarm by using

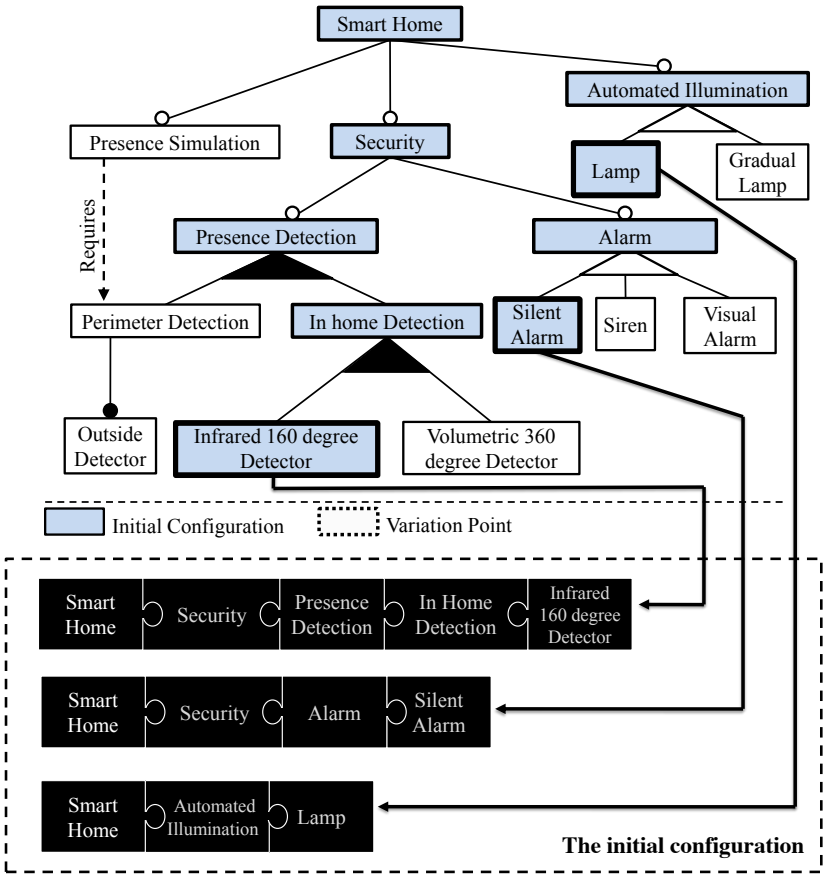


Figure 5.6: An example of the initial configuration of the smart home

the blink lighting.

Model To Text (M2T). Once the services have been configured, the transformation engine can be applied to generate the code. For this task, we have used the MOFScript language which provides capabilities for navigating models, creating files, etc. MOFScript takes as input one model and applies over one selected meta-element a contextual rule. The applied rule can access the element properties, navigate over the related model elements and invoke other rules.

The reader is referred to [148] for obtaining more information about the transformation rules and the tools to support the code generation.

Although this initial approach enables collaborative modeling by providing end-users with a closer language and achieves non-intrusive interoperability between PervML and the jigsaw metaphor, the expressivity of end-users is limited to the services that are previously designed in the feature model by the *Software Professional*. To overcome this limitation, we provided to advanced end-users with an open-option interface to describe new services. In this open-option interface, end-users are able to describe the needed information for a new service, which is: name of the service, where the service is located, what devices or services are needed to sense the context information (condition) and what devices or services are needed to activate this service (action). For example, end-users may need a climatization service that switches on the air conditioning when the window of the living room is closed and the temperature in the living room is over 26 degrees.

To support this in our tool, we have been inspired in existing EUD techniques such as *Natural Programming* and *Visual Programming*. In addition, the tool applies the following design interface decisions that were identified in Section 5.2: using a wizard, offering navigation buttons, displaying the elements using a grid layout, offering options, and using autocompletion. Only advanced end-users can use the open-option interface in order to reduce the complexity to non-advanced end-users. If a non-advanced end-user needs a service which is not included in the predefined catalogue, they need to interact with *Software Professionals* in order to describe the new requirements (new services and configurations).

The open-option interface offers advanced end-users the representation of the physical environment and a left frame that provides mechanisms to define the information required for a new service. Figure 5.7 shows a description of a new climatization service for the living room using the defined interface. The left side of the figure shows the required information: name, location,

condition and action. In order to introduce such information as location, end-users just need to select the corresponding option in the left frame and then select the location in the representation of the environment (at the right of figure).

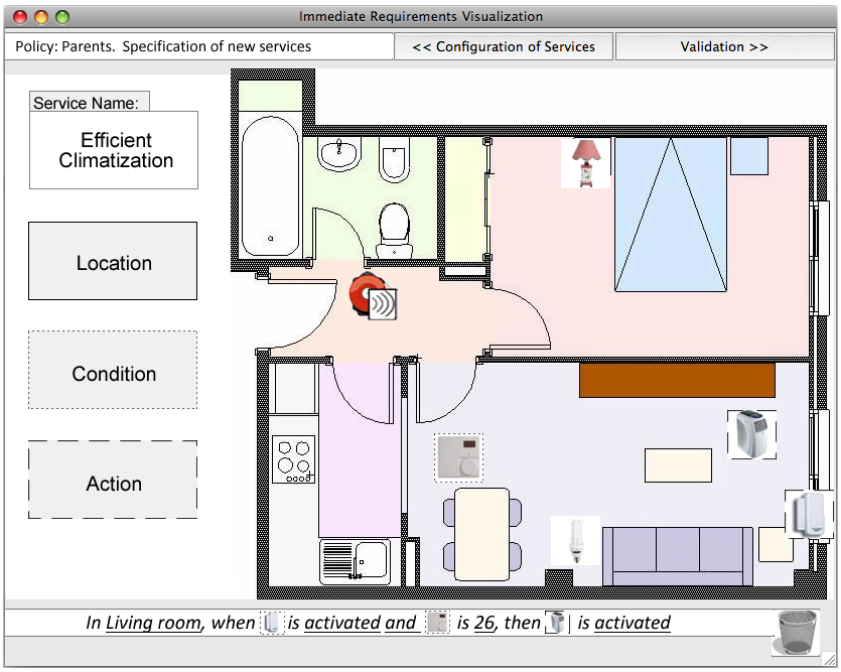


Figure 5.7: Open-option interface for describing a new service

In addition, the open-option interface shows in a visual way what information has been selected in the environment (right frame) and it also offers a text message where users can read and modify the result of their description (bottom frame). In this text message, some information can be modified such as: the value of the action or conditions (activated, inactivated or a numeric value) and how the language has to join two or more actions or conditions (and or or). This modifiable information is underlined in the text message.

To conclude, the advantage of using feature models is that model elements subject to variability are clearly marked but the initial configuration of

the system is limited to a bounded selection of features, which limits the expressiveness during the collaborative modeling. Nevertheless, although we propose an open-option interface to overcome the limitation of expressiveness for enabling advanced end-users to create new services, we still detect the following drawbacks and limitations in this approach:

- No integration mechanisms are provided if new services are created. This can cause conflicts during the integration of the new services with both the commonalities of the system and the initial configuration of the system using features.
- Two interfaces should be provided to users. One interface to support the selection of features and another one to create new services.
- There is no limitation about the concerns that users are able to modify during the creation of new services. Thus, users could modify commonalities of the system rather than be focused on describing the variants of the system. This may imply modifications of system descriptions that other users made, and it may create inconsistencies between the feature model and the description of new services.

To overcome our approach's shortcomings, we propose to use the facet of variability models rather than features for supporting collaborative modeling.

5.4 Conclusions

In this chapter, we have presented our proposal for achieving the involvement of end-users in modeling tasks of an existing MDD process for developing smart home systems.

Thus, end-users are able to share their knowledge with software professionals in order to enrich the system design since end-users are the ones who have

more in-depth knowledge about both the services that must be provided by the system and the environment in which the system is going to be deployed. In addition, the active participation of end-users in the system design creates on them a sense of ownership and minimize their mismatch between their expectations and system behavior [5], which makes results more difficult to reject in the future.

To start with, we have set the target of our approach by identifying the user skills and software activities. Afterwards, we present general guidelines and design principles that the EUD literature suggests to lower the barriers of users in the description of system behavior. In conclusion, users may be involved in modeling tasks using a modeling language that provides them concepts that they are familiar with. Although this work promotes the selection of existing modeling languages among the broad variety, it is not always available an existing modeling language that fits the goals and is closer to the users of a concrete project. In this case, we consider that the guidelines and design principles of the EUD literature that have been included in this chapter help the *Modeler expert* to design a modeling language that actively involves users in the modeling effort.

Finally, we have presented both our approach for supporting collaborative modeling using feature models and our tool-supported visual modeling language for involving end-users in the modeling tasks of an existing approach for developing smart home systems. Although this approach enables that end-users collaborate with software professionals in modeling tasks to obtain the initial configuration of the smart home system, this approach presented some limitations and drawbacks such as a bounded selection of features for end-users in the initial configuration of the system.

For this reason, it is necessary to evolve this approach for involving different types of users (e.g., scientists, engineers and end-users) by achieving

interoperability from different modeling languages and enabling collaborative modeling in modeling tasks of MDD processes. Thus, users describe themselves system properties that depend on them using a different modeling language that fits their goals, context and needs.

Chapter 6

ACHIEVING THE INVOLVEMENT OF USERS IN MODELING TASKS WITH HETEROGENEOUS MODELING LANGUAGES

In the previous chapter, we presented our approach for involving end-users in modeling tasks using a tool-supported visual modeling language. This tool allows end-users to select and customize system features using concepts that they are familiar with. However, giving end-users ways to easily customize behavior in well-specific domains, or customize their own tools to develop their daily work activities is important as we previously described; however, it is not enough [149], so it becomes necessary to empower different types of users with more expressiveness in order to support the development of a new generation of software systems. These systems need that different types of users are actively involved in modeling tasks to obtain a unified system description.

To address this, we explain in this chapter the sequence of steps that addresses the two last stages (*Specify and execute our proposal*, and *Build*

models in a collaborative way) of the collaborative modeling process that was explained in Chapter 4. We refer to this sequence of steps as the Medem method.

Medem enables the transformation and integration of descriptions in models from different modeling languages. Specifically, Medem supports the user of a modeling language to define a set of gaps in models, which are fulfilled by another user who describes them using a different modeling language. Thus, different types of users are involved and guided in the description of models using known concepts for them, which could help to adopt MDD processes by the software industry as expected [1]. Medem supports this collaborative modeling in a novel way by using the facet of modeling variability that uses models (which was introduced in Chapter 2). This facet provides Medem with mechanisms to manage the gaps.

This chapter is structured as follows: Section 6.1 presents our proposal for supporting collaborative modeling using models of a different modeling language. Section 6.2 presents the Medem method, which is divided into the specification of mechanisms (the specification phase) and the creation of models (the execution phase). Section 6.3 describes the steps that the *Modeler expert* performs in the specification phase of Medem. Section 6.4 explains the steps that users perform to build models in a collaborative way during the execution of Medem. Finally, Section 5.4 presents the conclusions of the chapter.

6.1 Supporting collaborative modeling using variability models

We identified in our initial approach, which was presented in the Chapter 5, new needs to support collaborative modeling from different types of users

since enabling them to select or customize system features in order to be involved in modeling tasks is not enough. Therefore, it becomes necessary to provide users with different mechanisms that 1) empower them with more expressiveness, and 2) provide them with integration mechanisms to describe the system features that depend on them in order to obtain a unified system description.

To address this, we propose to use in a novel way the variability management facet that uses models for managing the variability of products in a separate variability language, which was presented in Section 2.3. This facet could provide our approach with mechanisms to create variation points (as gaps to scope the participation of users), which can be described using models. Therefore, this facet could provide our approach the following benefits:

- It could increase the expressiveness of users.
- It stresses the importance of the involvement of users by describing their user-dependent properties using models rather than selecting features.
- It does not modify the structure of the modeling languages.

To support this facet, we choose the Common Variability Language (CVL) [90] because it pursues OMG standardization of variability modeling and management as we introduced in Chapter 2. Next, we present the current practice of modeling variability with CVL, and our envisioned approach using CVL.

Current practice of modeling variability with CVL:

CVL is based in the Base-Variation-Resolution approach (BVR-approach) [150] which argues to define orthogonal variability models that apply to a single base model.

The motivation of CVL is to separate variability modeling from the base domain modeling. CVL is suitable for modeling variability of models in any

base modeling language such as DSLs or UML. The CVL approach leaves the base domain modeling to the modeling language while the variability is treated with CVL (see Figure 6.1). This separation between the modeling language and the variability language provides a good separation of variability concerns. Thus, users who describe the base model can concentrate almost exclusively on the modeling language.

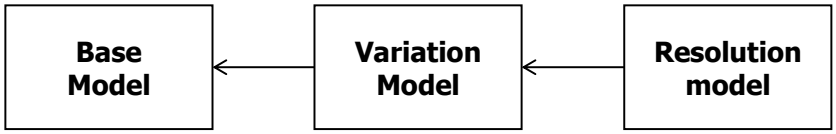


Figure 6.1: Base-Variation-Resolution Approach

As illustrated in Figure 6.1, a CVL specification consists of one variation-model that is applied to one base-model, and one or several resolution-models. The variation-model defines a set of alternatives (variabilities) for model fragments in the base-model. A fragment can be any arbitrary part of the base model, including a set of elements and their relationships. Finally, the resolution model determines the specific replacement choices for placements of the variability-model.

Figure 6.2 shows the current practice of CVL in which the different background colors highlight the CVL elements and the base model described using a DSL that is focused on a domain (see DSL_A in Figure 6.2). The figure also shows the transformations that CVL provides in order to obtain a *Resolved model*. The *Resolved model* is a DSL_A model in which the variabilities of the system has been fully described and all regular DSL tools (such as code generation) can be used.

It is important to highlight that the model fragments that describe the variabilities that could be identified in the base model are described using the same modeling language in CVL (see DSL_A in Figure 6.2).

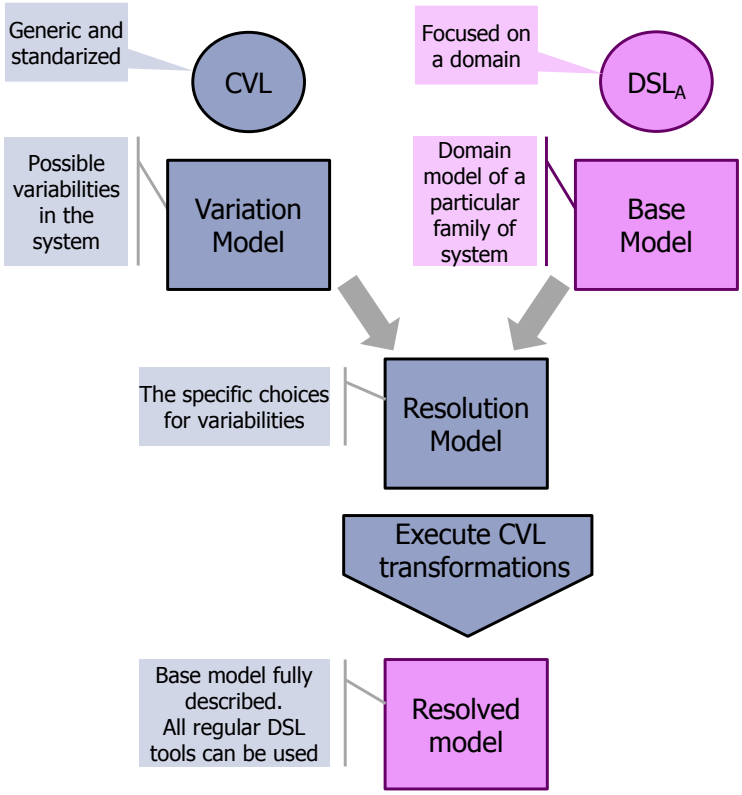


Figure 6.2: Modeling varibility with CVL

Our envisioned approach using CVL:

In our approach, we propose to use the variabilities that can be created in CVL to determine which model fragments of the base model can be refined using a different modeling language. Therefore, CVL provides our approach mechanisms to address the issue of scoping the user-dependent participation, which was identified in Chapter 4. Hence:

In our proposal, CVL is used in a novel way to specify not only variabilities in the base model but also use these variabilities to enable collaborative modeling from a different modeling language.

Figure 6.3 depicts our envisioned proposal for supporting collaborative

modeling from different modeling languages using CVL. The different background colors highlight the CVL elements and the two different modeling languages. On the one hand, the left side of the figure shows the current practice of modeling variability with CVL On the other hand, the right side of Figure 6.3 shows a different DSL, which may be closer to a different type of users (e.g., end-user) for supporting the collaborative modeling and involving them in modeling tasks (see DSL_B in the figure) by describing the variabilities. Specifically, the description of commonalities is performed using the DSL_A whereas the description of the variabilities is performed using the DSL_B. To support this, our approach uses interoperability mechanisms that transforms the description of variabilities from DSL_B models to DSL_A models. At the end, the CVL transformations are executed to obtain a resolved model that integrates system descriptions, which were made using a different modeling language.

Therefore, our approach supports collaborative modeling to actively involve and guide users in the specification of their user-dependent properties. The main advantage of our approach using CVL is that it 1) scopes the user-dependent participation by creating system variabilities, and 2) empowers the description of system variabilities using a different modeling language, and 3) integrates the description of the user-dependent properties to obtain a unified system description.

Our proposal can be applied in different domains, so it can enable collaborative modeling between models of existing modeling languages. To support this, we have defined a sequence of steps to initialize and execute the approach as well as supporting the construction of models in a collaborative way. These steps are described in detail in the next sections.

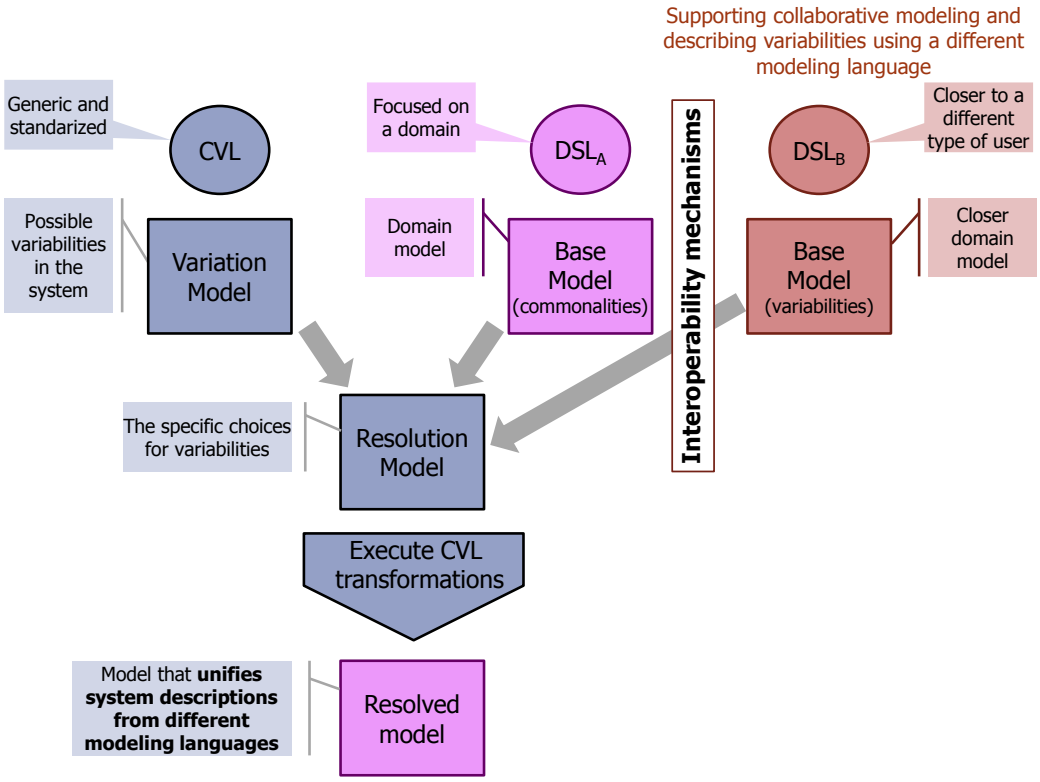


Figure 6.3: Our envisioned proposal using CVL

6.2 The Medem method

To tackle the issues (scoping the user-dependent participation and specifying user-dependent properties) that were identified in Chapter 4 to involve users in modeling tasks, and the envisioned approach that was previously presented, we propose the Medem method. Medem is comprised of two main building blocks that support model interoperability by means of model transformations, and variability management for enabling collaborative modeling using a different modeling language in a non-intrusive way with the structure of models.

Medem has 7 steps and it is divided into the specification of mechanisms

(specification phase) and the creation of models (execution phase). On the one hand, the specification phase is performed by a *Modeler expert* (Steps 1-3). On the other hand, the execution phase (Steps 4- 7) is performed between a user (e.g., a software engineer) who defines gaps as variable model subsets and another user (e.g., an end-user) who describes the gaps using a different modeling language. Thus, an unified system description can be obtained.

Figure 6.4 provides a general overview of Medem by showing its steps, involved artifacts and roles in the specification and execution phases. The shaded artifacts serve to show how Medem extends the hybrid approach for model transformations presented in Chapter 2 (non-shaded artifacts) by integrating variability management in a novel way for enabling collaborative modeling.

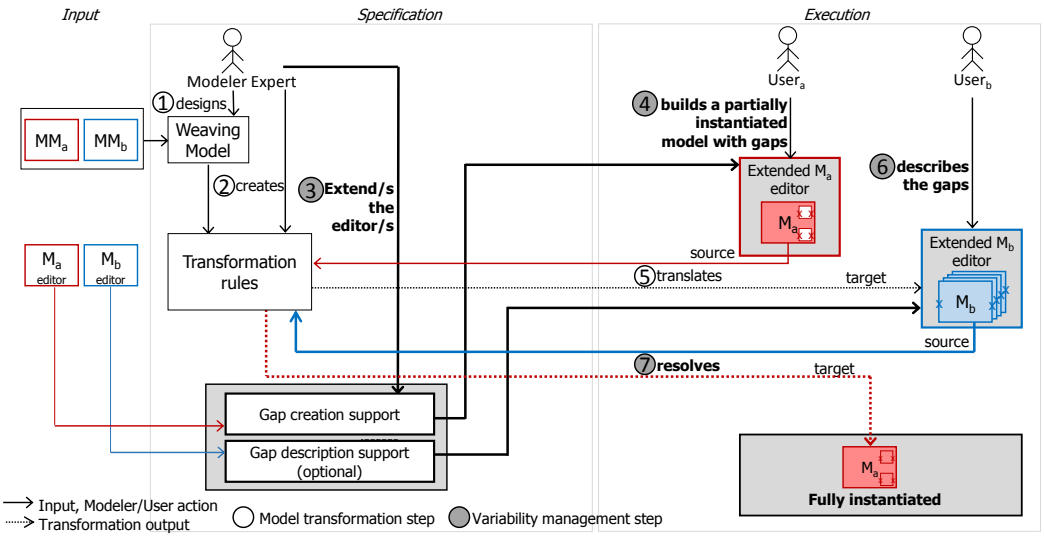


Figure 6.4: Steps of Medem during its specification and execution

Next, we describe each phase and step of Medem which each one is indicated in the figure as a number.

6.3 The Specification Phase

The aim of this phase is the specification of mechanisms that support the execution of Medem for enabling users to build models in a collaborative way. To start with, the *Modeler expert* determines whether the input meta-models (see MM_a and MM_b in Figure 6.4) of different modeling languages are able to exchange model descriptions (interoperable). To do this, the *Modeler expert* needs to: (1) be aware of the answer of questions such as [151]: *Do the models describe the same or different set of concerns? How are the models composed, and what are the relationships to sub-models?*, and (2) needs to overcome interoperability challenges since some concepts could have heterogeneities.

In this context, the use of model transformations is the cornerstone [22, 23, 24] of solving the connection among models of different modeling languages. As analyzed in the background in Section 2.2, this work is focused on a hybrid approach for model transformations since it is the most followed by the most adopted languages [22], and it becomes popular and useful tools in research and industry [71]. Moreover, this hybrid approach is non-intrusive with the structure of meta-models of the modeling approaches.

In this hybrid approach, the *Modeler expert* takes as input the meta-models for defining a weaving model with specific mappings (correspondences). This weaving model is used as input to create the model transformations. Nevertheless, the definition of the specific mappings and model transformations may present challenges to *Modeler experts* since structural heterogeneities can be found among bridged concepts in internal properties (such type, cardinality, etc.) and element relationships (such as inheritance). Moreover, semantic heterogeneities can be found among the concepts of the source and target meta-model (such as the target meta-model cannot always represent all the information from the source). These heterogeneities may

prevent the appropriate mapping specification and avoid the exchange of model descriptions between the modeling approaches.

Since the appearance of these challenges depend on the meta-models in which our approach is going to be applied and there is existing works (such as [76, 115]) that propose techniques to overcome structural and semantic heterogeneities, a detailed description of how to tackle them falls out the scope of the present work and relays on the *Modeler expert* knowledge.

In this work, the *modeler expert* determines as to whether or not are heterogeneities between the source and target meta-model. If so, the *modeler expert* determines whether the heterogeneities could be addressed by applying existing modeling techniques or model transformations. By contrast, the *modeler expert* determines that the heterogeneities cannot be addressed, so the appropriate definition of both weaving model and model transformations cannot be obtained. Therefore, the application of Medem is not feasible since the interoperability between models of the two involved approaches is not supported.

Moreover, the Modeler expert also determines the feasibility of Medem by taking as input the editors (M_a editor and M_b editor) that enable the creation of models, which correspond the MM_a and MM_b respectively, and determines whether the M_a editor can be extended to support the creation of gaps (variabilities) by selecting model elements. If the model editor is not able to support the creation of gaps by selecting model elements, Medem could be not feasible since it would be necessary that the user of the M_a editor creates gaps using a generic tree-like model editor and the variability management tool, which can be difficult for some users. By contrast, Medem is feasible.

Once the *Modeler expert* has determined that Medem is feasible, s/he performs the steps of the Specification phase (see the specification column of Figure 6.4): Step 1) designing a weaving model to link concepts between meta-

models, Step 2) obtaining model transformations according to the weaving model, and Step 3) extending the model editors to support Medem. These steps are described in detail below:

Step 1) Designing the weaving model. The weaving model is a model that contains different kinds of relationships to link the fullset of meta-model elements. To design the weaving model, we propose to use AMW as argued in Chapter 4. AMW consists in defining a specific mapping model (called weaving model) between the meta-model$_a$ and meta-model$_b$ (MM$_a$ and MM$_b$, respectively). The weaving model provides Medem with a bi-directional way to link elements of the meta-models involved. In addition, the weaving model is non-intrusive with the meta-models, so we do not need to modify the meta-models.

Step 2) Obtaining model transformations. The weaving model contains abstract and declarative links that are used to produce the fullset of model transformation rules. Transformation rules are used to enable interoperability between the model$_a$ and model$_b$ descriptions.

Note that according to [76] structural heterogeneities can be found among bridged concepts in internal properties (such type, cardinality, etc.) and element relationships (such as inheritance) and they can be resolved with transformation rules.

Step 3) Extending the model editors. The *Modeler Expert* extends the M$_a$ editor for supporting the creation of gaps whether s/he determined that the M$_a$ editor can be extended. To do this, the *Modeler Expert* implements an interface in this editor, which is explained in Chapter 7, to enable the creation of gaps (placement fragments in terms of variability concepts) as model$_a$ elements that will be involved in a substitution. The creation of gaps is carried out by selecting model$_a$

elements. A gap is defined by an identifier and a set of boundary elements that give the boundary between the gap and the rest of the model. The storage of gaps is automatically carried out in a variability model.

Although the M_b editor does not need to be extended, it can be a good option in some scenarios. For instance, if $User_b$ are end-users we may need editors that guide them in the description of gaps. To support this situation, specific tool support has been developed. It is explained in Chapter 7.

CVL [97] proposes two main concepts to express variability: placement fragment and replacement fragment. Table 6.1 shows the main Medem concepts, the CVL concept that supports them, and a brief description.

Medem	CVL	Description
Gap list	CVL model	The gap list contains the variabilities that has been identified and it is stored in a CVL variability model using an input $model_a$ (base model). The CVL model indicates which gaps must be described and how these gaps must be involved and related to other elements of the $model_a$
Gap	Placement fragment	Each gap described as variable in the base model by the $User_a$ is stored as placement fragment in the CVL model
Gap description	Replacement fragment	It contains model descriptions that fit in a gap

Table 6.1: Relation between Medem and CVL concepts

Therefore, CVL allows Medem to: (1) manage the placements of a base system model; (2) manage the components that can fit into the placements; (3) define a set of boundary points that give the boundary between a placement and the rest of the model; and (4) express gaps in a non-intrusive way with the structure of the m_a and the m_b models. Moreover, CVL provides tool support to display CVL concepts (such as placements) in a model editor. We identify each extended editor as *the Medem interface for $model_a$ or $model_b$* (from now onwards the extended $model_a$ or $model_b$ editor).

At this point, the specification phase has finished and its artifacts (the weaving model, model transformations and extended editors) are used in the execution phase in order to automatically achieve collaborative modeling from different modeling languages and obtain a unified system description. Moreover, it is important to highlight that these artifacts are reused in successive collaborative descriptions of the same modeling approaches.

6.4 The Execution Phase

In this phase, the Medem toolkit supports the creation of a fully instantiated $model_a$ that integrates $model_b$ descriptions in a transparent way for the users of the modeling approaches.

For exemplifying the execution phase of Medem, we use two well-known modeling techniques: Class Diagrams (CD) and Entity-Relationship (ER) diagrams. We select these modeling techniques as example because their concepts are well-know even though the prominent application of Medem is for enabling the participation of users who may be actively involved in modeling tasks of an existing MDD process using a different modeling language (e.g., involving end-users in modeling tasks of an existing MDD process for

obtaining an unified system description of a smart home system [152]).

Figure 6.5 shows the CD-ER example that aims the involvement of two domain experts who use different modeling languages, so they need work in a collaborative way to obtain a unified description of the system. As modeling domain, a simple warehouse information system is used. Next, we describe the steps of the execution phase of Medem.

Step 4) Building the partially instantiated model with gaps. The User_a reuses or builds from scratch a partially instantiated model_a using the extended model_a editor. Next, the User_a creates gaps in the model by selecting elements. Gaps will be described by the User_b using the model_b editor. The gaps are marked as variable in the base system model and they are automatically stored in the CVL model. Note that we use partially instantiated to denote that gaps are not described yet by a different user.

For example, Figure 6.5(1) depicts the CD model of the information system that a CD user has built. Furthermore, the CD user has created a gap to enable a different user (the ER user) to refine the description of how the sales are stored in the information system. The gap is represented in the figure by a grey square following the CVL concrete syntax [90]. The creation of gaps by selecting model elements is enabled in the extended M_a editor. How to extend this model editor is explained in Chapter 7.

For each gap, CVL creates a set of boundary points in which a crossed circle represents each one. Boundaries set the boundary between a gap and the rest of the model. In this example, the gap has a boundary in the *Sale* and *Product* classes.

Optionally in this step, the User_a can create replacement fragments

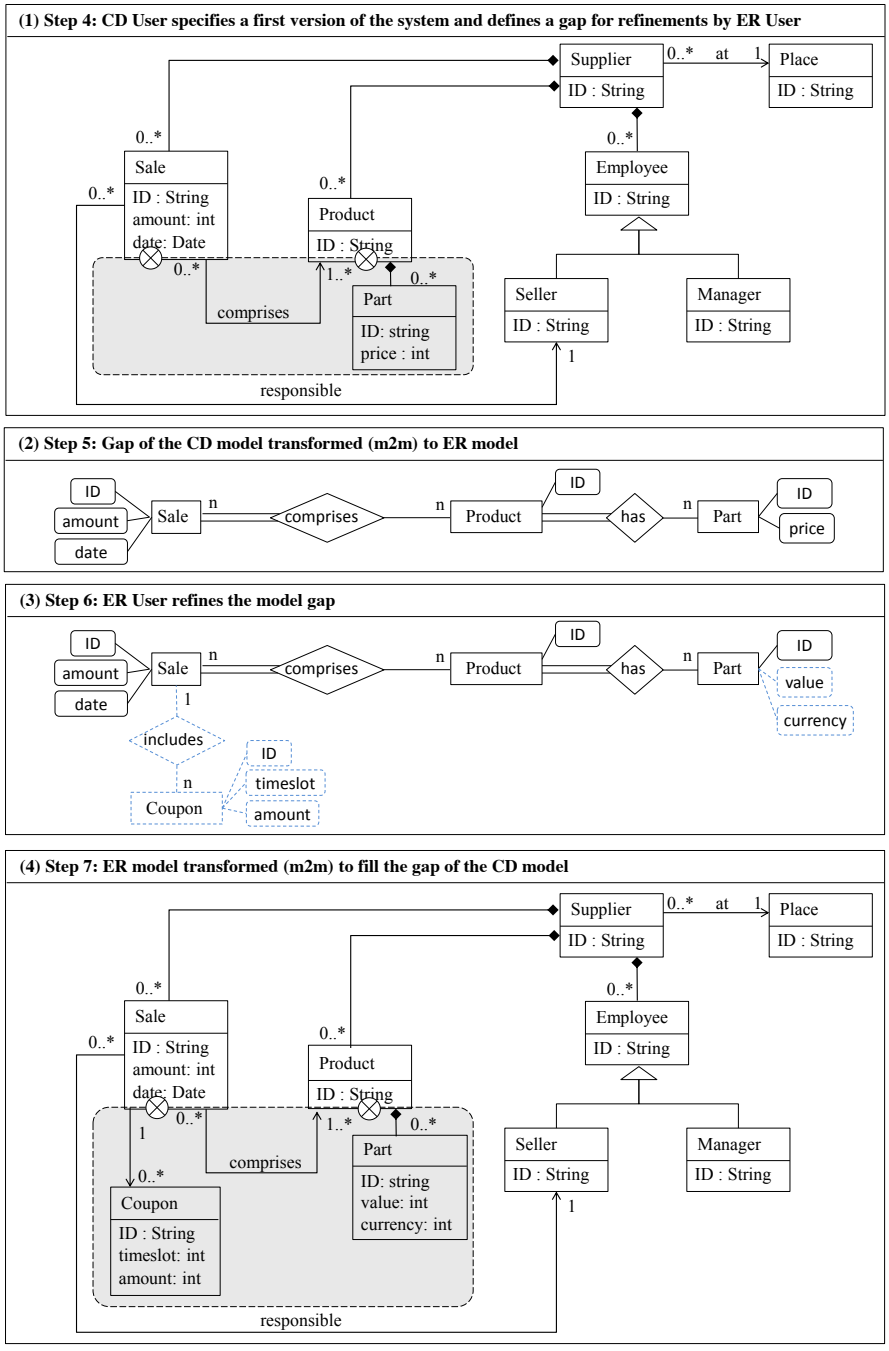


Figure 6.5: Example of collaborative modeling between class diagram model descriptions and relationship model descriptions

(predefined descriptions) for each gap to provide users with a library of components that help their involvement in modeling tasks and make the participation of users easier according to the guidelines identified in Section 5.2. To do this, model elements can be selected in the base system model for creating a replacement fragment from this selection.

Step 5) Translating to support gap descriptions. The transformation rules, that were created in the Step 2, takes as source the partially instantiated model_a to automatically translate the content of each gap into a target model_b. Moreover, the elements involved as boundaries for each gap are also translated for showing them using model_b concepts. For example, once the CD user ends the creation of gaps, m2m transformations are executed to translate each gap and its context (the boundary model elements) to a ER model. Figure 6.5(2) depicts the gap and its context transformed in a ER model that the ER user can refine.

Step 6) Describing the gaps. The User_b refines the gaps using the M_b editor or its extended version. The M_b editor stores each gap description in a model_b transparently to the User_b. For example, Figure 6.5(3) shows the refinements that the ER user has performed: a new entity to manage coupons in sales and a refinement of the *price* attribute, which has been changed to to store the value and currency (see the refinements denoted by dotted blue lines in the Figure 6.5(3)).

Step 7) Obtaining the fully instantiated model_a. A process automatically resolves the gap descriptions into a resolution model, which is a model_a. Thus, this resolution model combines descriptions that have been performed in M_a and M_b by the User_a and User_b respectively. The process is made up of the following steps: 1) the transformation

rules takes as source each gap description in a model_b to automatically translate it into a model_a, 2) the variability model is updated to include each gap description as a replacement fragment and create resolution elements, and 3) an automated mapping provided by CVL transforms the variability model into the resolution model_a. The resolution model uses the concepts of the base language (model_a). Thus, the partially instantiated model_a that is taken as input at the beginning of the process is enriched with model_b descriptions.

For example, once the ER user has finished the gap refinements, both a m2m transformation from the ER model to the CD model and the transformation that is provided by CVL are executed to obtain a fully instantiated CD model. This fully instantiated model is shown in Figure 6.5(4) in which the gap is fulfilled according to its ER model description.

At this point, successive collaborative modeling descriptions can be automatically supported by returning to Step 4. For example, the CD user creates new gaps to enable the ER to refine the description of different concerns of the system.

6.5 Conclusions

In this chapter, we have presented our Medem approach for supporting collaborative modeling from two different modeling languages using variability models. To achieve this, we have described the steps that are necessary in the specification and execution of Medem. These steps combine modeling and variability techniques that achieve non-intrusive interoperability between model descriptions of two different modeling languages. In addition, the guidelines of the EUD literature that have been presented in previous chapters can be followed to provide different types of users with a closer different

modeling language, whether there is no modeling language that fits the needs of users of a concrete project.

To build the sequence of steps, we have tackled the issues that were identified for involving users in MDD processes (scoping the user-dependent participation and specifying user-dependent properties), we have overcome the key and critical challenges that have been identified for achieving collaborative modeling, and we have taken into account the lessons learned in the collaborative modeling field. For example, one design decision that has been applied is the use of variability management in a novel way to specify not only variabilities in the base model but also, use these variabilities to determine which model concerns have to be described with a different modeling approach.

The main innovation of Medem is the combination of both variability and modeling techniques to 1) scope the user-dependent participation by describing gaps and completing them, and 2) reuse the Medem initialization if new model descriptions are carried out. Furthermore, we believe that using our model-driven and variability-based approach is a promising way to integrate model descriptions that have been performed using a different modeling language. This approach brings the following important benefits: 1) users are able to participate in the modeling effort of an existing MDD process using a modeling language that they are familiar with; 2) different modeling languages are able to interoperate in order to obtain the full description of a software system; 3) users can be focused on describing the concerns of the software system that they are experts with rather than describe the entire software system; 4) the structure of the modeling languages is not modified, which promotes the application of the approach in existing modeling languages; and 5) it could help to improve the adoption of MDD processes.

The next chapter shows the technological decisions to support the princi-

pal building blocks of Medem (interoperability and variability mechanisms).

Chapter 7

MEDEM TOOL SUPPORT

In this chapter, we describe the technological decisions that support the principal building blocks of Medem (interoperability and variability mechanisms) in order to create a toolkit that embeds the development of these building blocks. Specifically, we have implemented a toolkit prototype that entails a variability model to manage gaps, and a weaving model and transformation rules to support model-to-model transformations. The toolkit supports collaborative modeling from different modeling languages in a transparent way once it is initialized at the beginning and this initialization can be reused in different projects of the same modeling languages.

The technological decisions that support the toolkit are aligned with current modeling standards and MDD-oriented technologies. Therefore, we favor that researchers and practitioners apply our approach to other existing modeling approaches. In particular, we have proposed MOF [63] meta-models as stating point in our approach to support non-intrusive interoperability of any DSL based on MOF and more specifically, the Eclipse implementation of it (the Eclipse Modeling Framework (EMF) [153]).

Also, as stated in previous chapters, we have proposed CVL variability

models (a proposal sent by IBM, Thales, Fraunhofer FOKUS and TCS for the Object Management Group) to support collaborative modeling. In order to use our toolkit as is, it requires that the meta-models of the modeling languages are defined using EMF, and the model editors store model descriptions using XML Metadata Interchange (XMI). Moreover, it is recommended that the M_a editor can be extended to implement an interface that is provided by the CVL Tool for creating gaps by selecting model elements. For further details on these technologies and the CVL Tool we refer the reader to [153, 154, 155].

Figure 7.1 gives an overview of the technological decisions that support the main building blocks of our approach. In addition, the figure shows how these technological decisions are related to the steps (corresponding to the different numbers shown in the figure) to specify and execute Medem. To support the specification of model transformations between $model_a$ and $model_b$ in Steps 1 and 2, the AMW and ATL eclipse plugins are proposed to be used. To extend model editor in Step 3 for supporting variability, CVL and its associated tool support is used. To manage CVL gaps and integrate them into a final model in Steps 4-7, we use EMF model queries, m2m transformations, and a CVL transformation. Steps 4 and 6 use model editors and do not need specific tool support.

This chapter is structured as follows: Section 7.1 describes the technological decisions for supporting model transformations in Steps 1-2 of Medem. Section 7.2 shows the support of variability management in Step 3. Section 7.3 describes the mechanisms for supporting integration of models in Steps 5 and 7. Section 7.4 shows an example of usage integrating ER and CD model descriptions. Finally, Section 7.5 concludes the chapter.

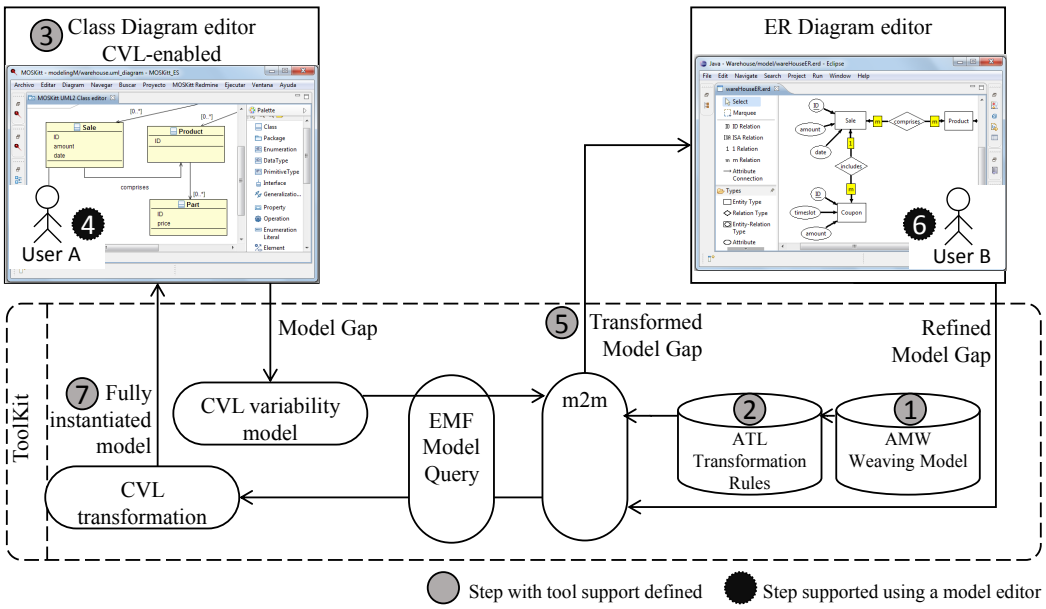


Figure 7.1: Overview of the technological decisions that support the steps of Medem and how they are related

7.1 Supporting model transformations in Steps 1-2

For enabling the toolkit to support the model transformations, the weaving model and transformation rules have to be provided by the *Modeler expert* in Steps 1 and 2 of Medem (see 1 and 2 in Figure 7.1).

To create and handle the weaving model, we propose the use of the ATLAS Model Weaver (AMW) [156] Eclipse plugin. The AMW tool provides a set of standard facilities for the management of weaving models and meta-models [22] such as an easy-to-use and intuitive editor for conforming models. Specifically, this tool allows *the Modeler expert* to establish relationships (i.e., links) between the Meta-Model$_a$ and the Meta-Model$_b$ to store them

in a model named *weaving model*. Moreover, we propose the ATLAS Transformation Language (ATL) [157] for the development of the model transformation according to the links of the weaving model. ATL is the preferred option in the community [22], it is considered the *facto* standard for the development of model transformations [39] and it is coupled with the AMW tool.

Figure 7.2 shows a snapshot of a fragment of the weaving model to support the CD-ER example that was described in the previous section. The left-hand panel shows the meta-model concepts of the CD (represented in the EMF tree-like editor) and the right-hand one shows the meta-model concepts of the ER. The center panel shows the weaving model and the links among concepts (correspondences) that were created by the *Modeler expert*.

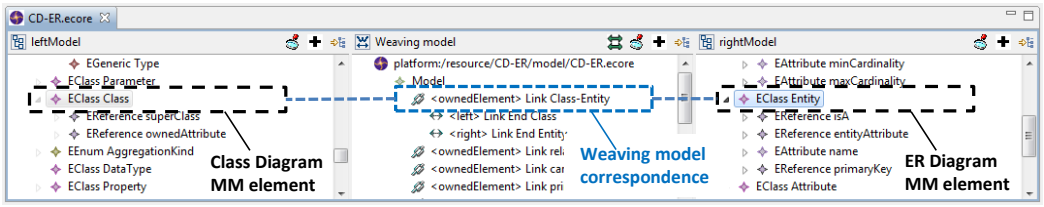


Figure 7.2: A fragment of the weaving model for the CD-ER example

Once the weaving model is designed, the AMW tool could interpret the weaving model to transform the links of the weaving model into an executable ATL transformation as output [76]. This output will be used by the toolkit to enable interoperability between $model_a$ and $model_b$ descriptions.

7.2 Supporting variability management in Step 3

For supporting variability management, the M_a editor may be extended to enable the automatic creation of gaps in the CVL variability model by selecting $model_a$ elements. To formalize the creation and description of gaps, we use concepts that CVL [155] proposes to express model variability such as *PlacementFragment* (a set of base model elements that can be replaced), *ReplacementFragment* (a set of model elements that will be used as replacement for some placement fragment), and *Boundary* (model element that represents the boundary between a placement fragment and the rest of the base model).

For extending the M_a editor to create gaps, CVL provides tool support that includes an API [155], which can be implemented in a base model editor to become CVL enabled. In particular, the interface *ICVLEnabledEditor* provided in the CVL API should be implemented [155] to obtain the CVL enabled version of the M_a editor. The CVL editor interoperates with any editor implementing this interface in which its meta-model is defined using EMF (specifically, an ecore meta-model). In total the interface counts 4 operations. This interface allows getting and setting the base model elements selected in the M_a editor as well as highlighting $model_a$ elements. The selecting capabilities are used to both make selections and links to the $model_a$ elements and build fragments from selections in the CVL variability model, and the highlighting capabilities allow displaying the variability in the M_a editor.

Once the M_a editor is CVL enabled, the resulting behavior for the user consists of selecting the model elements that are going to be included in the placement fragment, and choosing ”create placement fragment from selected”

from right-click pop-up menu. This behavior is automatically achieved in the M_a editor with the implementation of the CVL interface described above. The *PlacementFragment*, the *ReplacementFragment* and the *Boundary* points will be generated automatically in the CVL variability model. Whether the M_a editor could not be CVL enabled, the CVL tool enables the creation of the CVL variability model using a generic tree like model editor but this can be difficult for M_a users, so the *Modeler expert* will determine its feasibility as explained in the previous section.

By following the CD-ER example, the upper part of Figure 7.3 shows the CVL-enabled editor to support the automatic creation of gaps by selecting CD model elements and choosing "create placement fragment from selected" from right-click pop-up menu.

Supporting the optionally extension of the M_b editor

Although the M_b editor does not need to be extended, it can be a good option in some scenarios (e.g., $Users_b$ are end-users who need editors that guide them in the description of gaps). Hence, specific tool support has been developed to guide the description of gaps.

The *Modeler expert* could extend the M_b model editor by adding the following areas: *Components* (shows a guide with all the placement fragments of the variation model that should be described), *Components personalization* (enables the description of placement fragment using m_b concepts and highlight the m_b concepts that are boundaries), *Information* (helps with the description of components), *Library of components* area (shows the available replacement fragments for a placement fragment) to provide a library of components as a starting point to refine a component. Thus, a placement fragment can be refined with a replacement fragment from the library of components in the extended M_b editor, or it can be refined using m_b concepts.

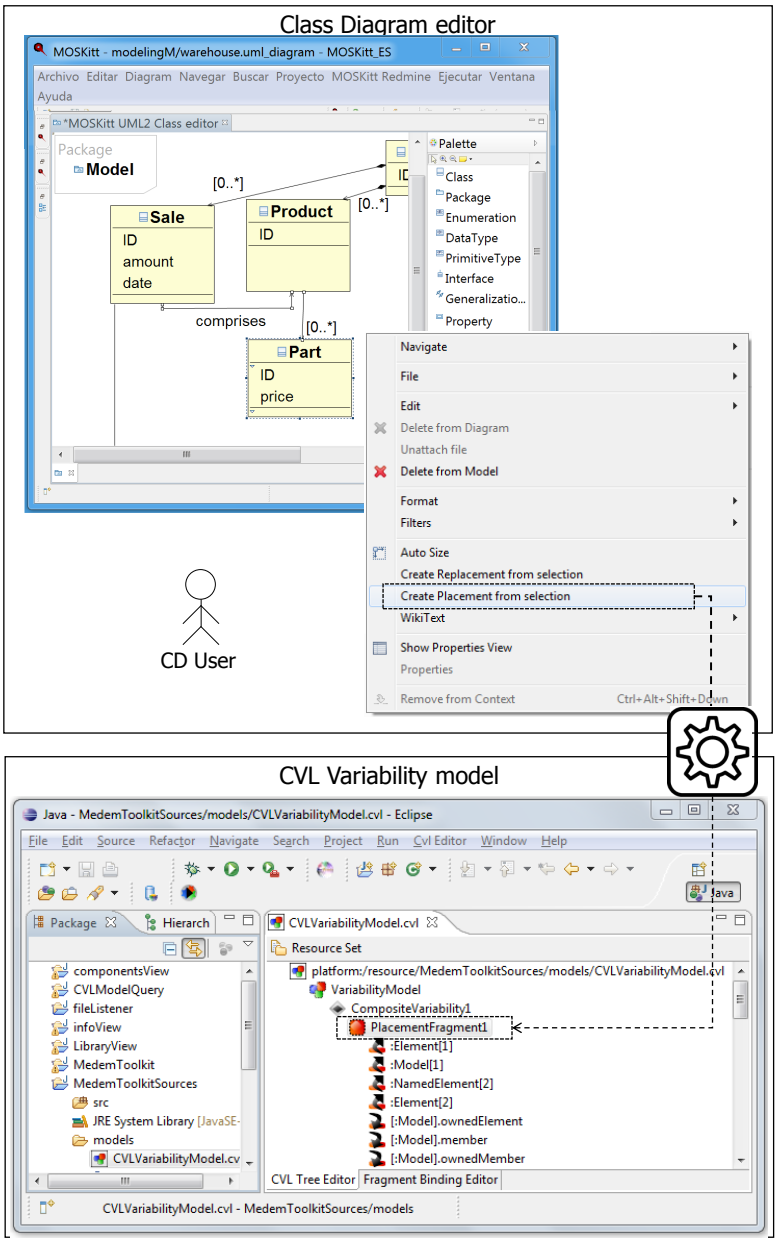


Figure 7.3: Snapshot of CVL variability model once a gap is automatically created by selecting model elements in the CD editor

To fulfill the *Components*, *Information*, and *Library of components* areas, we use the EMF Model Query framework (EMFMQ) [158]. To complete the components area, we use the EMF Model Query framework (EMFMQ) [158].

EMFMQ provides an API to construct and execute query statements in a SQL-like fashion as follows:

```
1  SELECT
2  FROM modelElements
3  WHERE condition
```

These query statements are included in the toolkit for discovering and modifying model elements. Queries are first constructed with their query clauses and then they are ready to be executed. There are two query statements available: SELECT and UPDATE. The SELECT statement provides querying without modification while the UPDATE statement provides querying with modification. Every query statement requires some query clauses. The SELECT statement requires two clauses, a FROM and a WHERE. For example, we initialize the *Components area* of the extended model_b editor by executing a domain-independent query that we have implemented [152] using EMFMQ, which is for selecting all the placement fragments of a given resource (the CVL variability model).

At this point, it is also worth pointing out the importance of integrating model validation mechanisms in the toolkit since we detected some problems during gap descriptions (i.e., gap descriptions do not fit the boundaries in the variability model and a component is described in a gap that was already described). To address this, the toolkit executes a set of EMF Model Queries in gap descriptions, the weaving model, and the partially and fully instantiated model_a to show the result in the *Information* component that could be added within this extension of the M_b editor. For example, we define a rule that checks if gap descriptions satisfy the boundary elements in the CVL model. If some model_b description does not satisfy the boundaries elements, a message is shown in the model_b extended editor such as: "The *model_b concept* has to be included in the description of the component *name*".

7.3 Supporting integration of models in Steps 5 and 7

In Step 5, the toolkit executes for each gap the ATL model transformation to obtain a M_b model that the $User_b$ could refine. Specifically, the toolkit manages both the CVL variability model using the CVL tool support and the model transformations to transform the gaps from one modeling language to another.

In addition, we defined generic model queries in the toolkit to obtain information of the CVL variability model (i.e., a list with the gaps that should be described). For example, the select statement to get the list of the *PlacementFragments* of the CVL variability model is as follows:

```
1 SELECT statement =
2  new SELECT(
3   new FROM(resource.getContents()),
4   new WHERE(new EObjectTypeRelationCondition(
5    CvlPackage.Literals.PLACEMENT_FRAGMENT,
6    TypeRelation.SAMETYPE_OR_SUBTYPE_LITERAL));
```

In Step 7 of Medem, the toolkit transforms each refined M_b model to a M_a model and updates the CVL variability model to obtain the fully instantiated model, which combines descriptions obtained from the both editors. In particular, it performs the following as described in the previous section: 1) translates the $model_b$ concepts into $model_a$ concepts using the transformation rules, 2) stores the information in the variability model using EMFMQ update queries, and 3) executes a CVL transformation to obtain the resolution model.

For example, we have implemented an *update* statement by using EMFMQ to update a replacement named *ReplacementFragment1* to store the descriptions (*ElementsFromDescription1*) in a given resource (the CVL

variability model). The update query is as follows:

```
1  UPDATE statement =
2    new UPDATE(
3    new FROM(resource.getEObjects()),
4    new WHERE(new EObjectAttributeValueCondition(
5      CvlPackage.eINSTANCE.getCVLNamedElement_Name(),
6      new StringValue(ReplacementFragment1))),
7          new SetRFragment(ElementsFromDescription1));
```

Figure 7.4 shows a snapshot of the Medem toolkit once it completes a gap description (e.g., how the sales are stored in the information system) in the CVL variability model using model transformations and update EMF model queries. In particular, the figure shows a subset of the CVL variability model that has one gap (the red square icon), a gap description (blue and red icon), and a resolution element (green icon) that links the gap with its description. Moreover, the figure shows the Medem toolkit log that shows information throughout the process (such as the elements that were obtained using the EMF model queries).

Once the refinements of gaps has been finished, the fully instantiated model (the resolution model in terms of CVL concepts) is obtained as a result taking the CVL variability model as input and using the CVL resolution model generator [97]

7.4 Example of usage: integrating ER and CD model descriptions

For executing and using the toolkit (Steps 4-7 of Medem), the *Modeler expert* has to provide both the weaving model and the ATL model transformation previously described. Then, the toolkit automatically supports collaborative

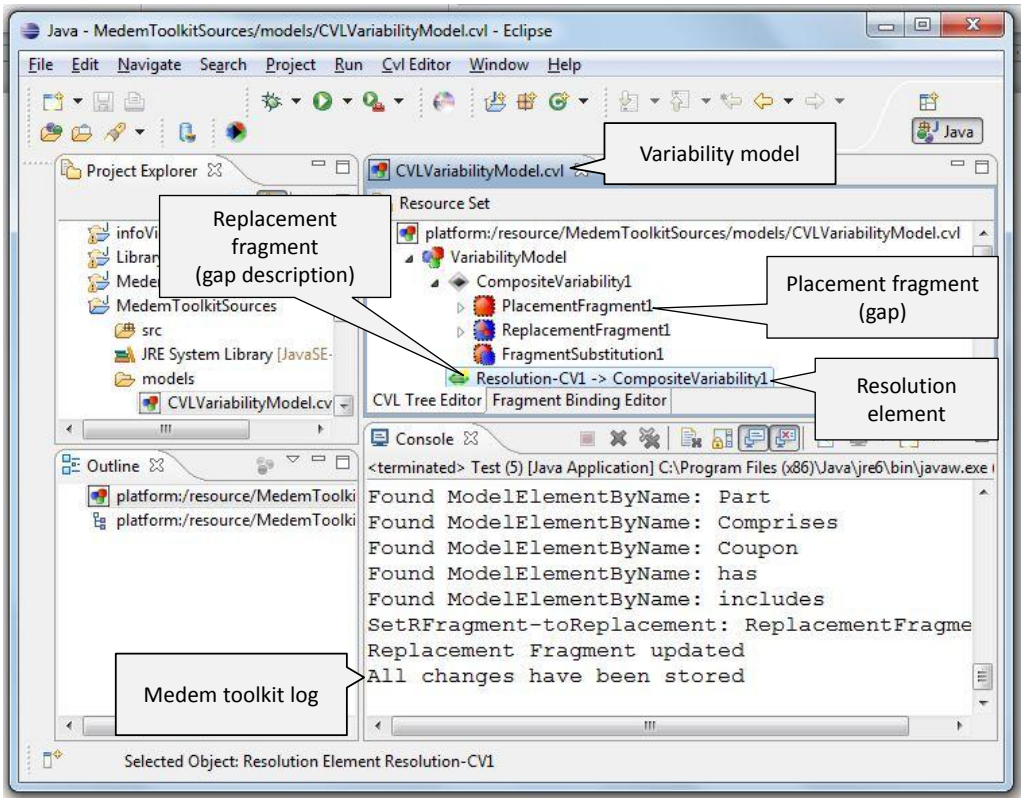


Figure 7.4: Snapshot of the Medem toolkit prototype

modeling descriptions by creating (Step 4) and describing gaps (Step 6) from a different modeling language.

By following the CD-ER example, the lower half of Figure 7.3 shows a snapshot of the CVL variability model with the *PlacementFragment* (the red square icon) with its boundaries (arrow icons) that has been automatically created once the CD user selected model elements in the CVL-enabled Class Diagram editor in order to allow a different user (the ER user) to refine the description of how the sales are stored in the information system.

After, the toolkit automatically provides the ER user with a ER model for each gap to enable its refinement. Once the ER user has finished the

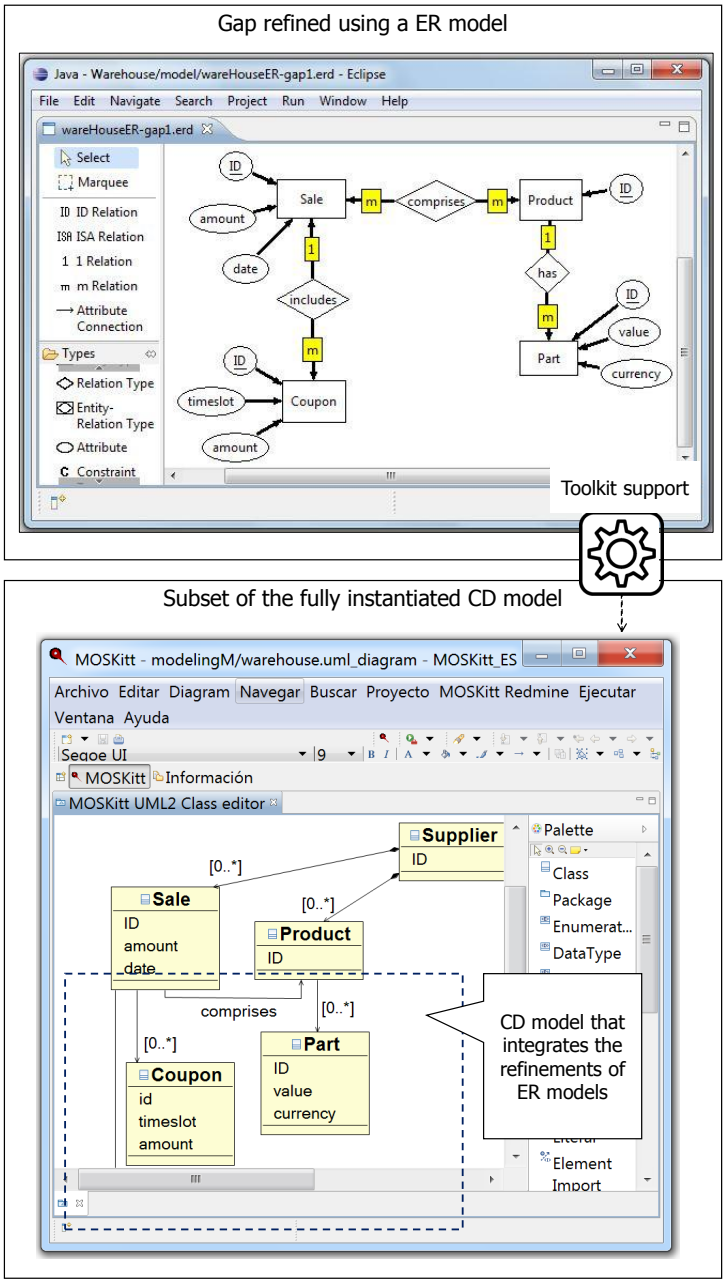


Figure 7.5: Integrating ER and CD model descriptions

refinement of ER models (see an example of gap refined using a ER model in the upper half of Figure 7.5), the toolkit automatically follows the process that was described in Step 7 to automatically obtain a fully CD model that

integrates the refinements of ER models as the lower half of Figure 7.5 shows.

Finally, we would like to highlight that the toolkit automatically reuses the weaving model and the ATL transformation in successive collaborative descriptions, which comprise meta-model concepts that have already been used. For example, the ER user could change the attributes of the *Coupon* entity that is shown in the upper half of Figure 7.5 in successive collaborative modeling descriptions. In that case, the toolkit will automatically support that change without requiring the *Modeler expert* to define correspondences again.

7.5 Conclusions

In this chapter, we have presented the technological decisions to support our Medem approach. The technological decisions that support the main building blocks of our approach (collaborative modeling and interoperability mechanisms) are aligned with current modeling standards and MDD-oriented technologies such as MOF meta-models and CVL variability models.

The main advantage of our Medem approach using CVL is to empower users as an active role in the modeling effort of the system properties that depend on them using a different modeling language that fits their context and needs. Therefore, Medem offers a balance between keeping the properties of m_a models and having the flexibility to describe properties using m_b models. Medem also uses an API to build and execute query statements for discovering and modifying elements of both the weaving model and the CVL variability model. We would emphasize that both the weaving model and the CVL variability model are non-intrusive with the meta-models of the modeling languages.

Although our approach automatically supports collaborative modeling

descriptions in a transparent way for users once it is specified, the specification of the approach by the modeler expert may require the most time and effort. On the one hand, selecting the proper modeling approaches among the broad variety, especially if it is not available an existing modeling approach that both fits the project goals and is closer to the users that may be involved.

On the other hand, we have detected that the first three steps of Medem (designing the weaving model, obtaining the model transformations and extending the model editors) could require more time than the remaining steps of Medem. However, this phase is reusable even though users change model descriptions or create new ones. Moreover, this initial time could be reduced using existing approaches such as [76] to accelerate the development time of the weaving model and model transformations.

Since the technological decisions that support the toolkit are aligned with current modeling standards and MDD-oriented technologies, researchers and practitioners will be more inclined to apply our proposal to other existing modeling approaches. Nevertheless, we believe that these decisions could present some limitations in the application of our approach in some cases (e.g., users that use a language that is not based on MOF, or some required inputs to apply our approach such as the Ecore meta-models of the involved modeling approaches are not provided).

The next chapter shows the application of Medem in three case studies of different domains in order to evaluate their applicability and feasibility.

Chapter 8

EVALUATION OF THE PROPOSAL

This chapter describes the application of our proposal in practice for evaluating its applicability and feasibility. Medem was applied in three case studies of different domains (smart home systems, web information systems, or biomechanical protocols) that show the necessity of enabling different types of users (end-users and software professionals; domain experts and software development experts; and doctors and biomedical engineers, respectively) to collaborate in modeling tasks.

In order to perform the case studies, we follow the guidelines provided by Runeson and Höst [126]. Thus, we design and plan the three case studies in which each one takes as input an existing MDD process of a different domain that does not involve different types of users in modeling tasks even though they should be actively involved in the description of domain-specific content.

For each case study, we present its highlights and its applicability of Medem. In particular, we describe for each case study the following: 1) why Medem was applied, 2) the involved DSLs and a snapshot to provide an overview, 3) an example of a correspondence designed in the weaving model to show an example of transformation between concepts in the case study.

Finally, we conclude by discussing the results. The results have proven the applicability and feasibility of our approach in a non-intrusive way with the structure of models of existing MDD processes. In addition, the results serve to discuss not only the advantages of Medem but also, its drawbacks and future improvements.

This chapter is structured as follows: Section 8.1 presents the case study of smart home systems. Section 8.2 shows the case study of web information systems. Section 8.3 describes the case study of biomechanical protocols. Finally, Section 8.4 discusses the main conclusions of the results obtained from the application of the three case studies.

8.1 PervML - Pantagruel Case Study

We evaluated our method by applying it in an existing MDD process that address the development of pervasive systems [152]. As was introduced in Chapter 5, this process is PervML and provides a DSL that is focused on specifying heterogeneous services in concrete physical environments such as the services of a smart home. PervML has been successfully applied to develop solutions in the smart home domain [99]. The main characteristics of PervML are the following:

- The services, devices, and locations of the smart home are described using a graphical syntax based on UML.
- The service and device behaviors are specified using the Action Semantic Language (ASL).
- The triggers of the services are specified using the Object Constraint Language (OCL).

- Six models are required to specify the above information. More detailed information about these models can be found in [99].

Although the configuration of the smart home system relies on end-users' descriptions, end-users cannot participate in the description of their smart home system because end-users lack the skills to manage the technologies that PervML uses (i.e., ASL, OCL, or UML) and describe the smart home system using concepts that end-users are not familiar with (i.e., component, trigger, interaction, or binding provider) in different models. For example, the upper half of Figure 8.1 shows a snapshot of six PervML models that are required to describe a smart home. These models describe services (i.e., comfort and security services), devices (i.e., alarm and presence detector devices), and the location of the services and devices (i.e., living room, bedroom locations) of the smart home. These models not only describe the smart home services but also the available devices, operations and behavior.

As end-users need a language that uses concepts familiar for them in order to collaborate in the PervML modeling phase, we apply Medem. Thus, the smart home system description is completed using model descriptions of another existing DSL called Pantagruel [159]. Pantagruel integrates an end-user oriented language to describe a pervasive environment following a sensor-controller-actuator development paradigm. This paradigm improves the usability of non-professional participants according to the studies assessed in [159]. Pantagruel offers users the following:

- Expressiveness to describe the events of the smart home system.
- Improvements in the development process using its visual programming language. This is due to the representation of objects using spatial relationships. Pantagruel offers a spatial structure of entities, rules, sensors, and actuators that could be customized.

- Improvements in the abstract and concrete syntax with regard to PervML. For example, two concepts of PervML are: Device and Service. By contrast, Pantagruel does not have the concept of Service. Pantagruel links devices that can work as a sensor or actuator to design the events of the system. These links are made by means of visual representations. This makes Pantagruel more intuitive for users than PervML (i.e., sequence diagrams are used in PervML to design the events of the system).

Therefore, we apply the steps of Medem to support collaborative modeling between PervML and Pantagruel.

First of all, the *Modeler expert* designs the weaving model to bridge all the PervML and Pantagruel concepts (Step 1 of Medem). The PervML meta-model has about 67 elements among meta-classes, classes and attributes and the Pantagruel meta-model has about 28 elements. These meta-models are taken as input to design correspondences in the weaving model. For example, the weaving model bridges the (*Device, Service*, and *Trigger*) PervML concepts and the *Sensor* Pantagruel concept. Note that, there are Pantagruel concepts that have multiple correspondences with PervML concepts and vice versa. After, the *Modeler expert* obtains the transformation rules according to the weaving model (Step 2 of Medem).

Next, the *Modeler expert* extends the PervML and Pantagruel editors to support the gap creation and description, respectively (Step 3 of Medem). Then, the execution phase of our approach starts and the *PervML analyst* builds a partially instantiated PervML model (Step 4 of Medem) with gaps (i.e., a gap to allow end-users to describe how the comfort service in the parents room works). These gaps are translated (Step 5 of Medem) to be completed by end-users in the Medem extended editor for Pantagruel (Step 6 of Medem) as the snapshot of the lower half of Figure 8.1 shows (i.e., the

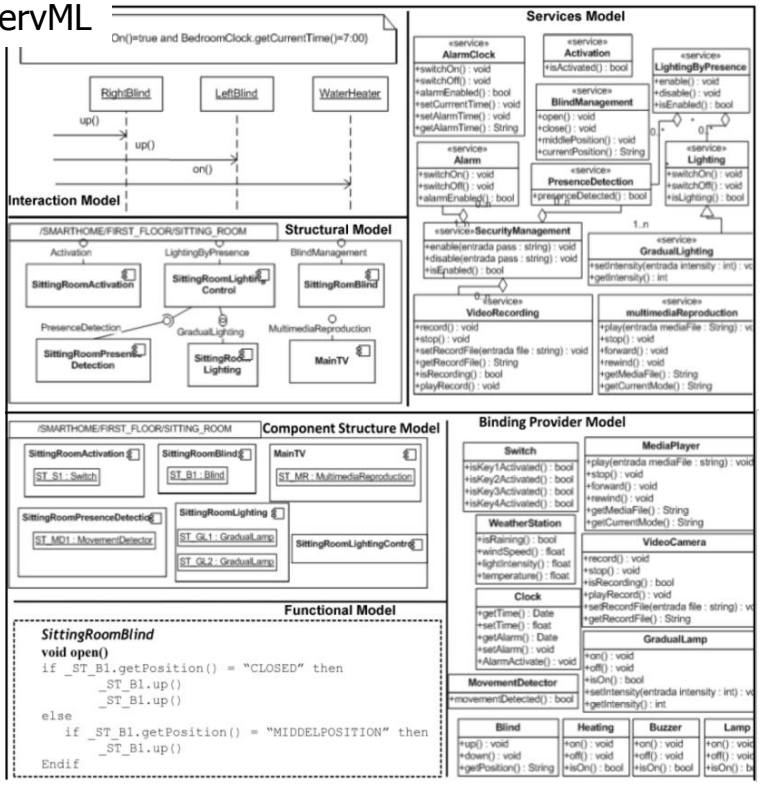
ParentsRoom comfort gap is described to set the blinds of the parents room to the middle position, switch off the garden lights, and set air conditioning temperature to 20 when the activator of the parents room is enabled). Once end-users complete the descriptions of gaps, the Medem toolkit automatically translates Pantagruel model descriptions to PervML descriptions (Step 7 of Medem) to resolve a fully PervML instantiated model.

The result was that end-users can actively participate in the description of concerns of their smart home, and their descriptions were integrated to obtain a unified system description in PervML models. Thus, the end-users' involvement that Medem provided does not force to end-users neither manage the 6 different PervML models that describe a smart home system nor have knowledge about the technologies required to manage these models.

Therefore, as the application of Medem combined two existing modeling languages in a non-intrusive way to obtain a common system description, it not only inherited their advantages (i.e., the expressiveness that the Pantagruel provides to end-users) but also, it added the following advantages: (1) the PervML models were not only designed by the software professionals, the PervML models were also enriched by the end-users; and (2) end-users were provided with examples that help them in their involvement and organization of their descriptions.

In the application of this case study, we detected that the gaps that software professionals created in PervML to involve end-users in modeling tasks seek to orchestrate different services in order to fit end-users' preferences and they are not involved in the description of other concerns of the system (i.e., how the devices and operations work). For this reason, it was not necessary the fullset of correspondences in the weaving model and transformation rules that were provided at the beginning by the *Modeler expert*. The *Modeler expert* should be provided with mechanisms that

A) PervML



B) Medem extended editor for Pantagruel

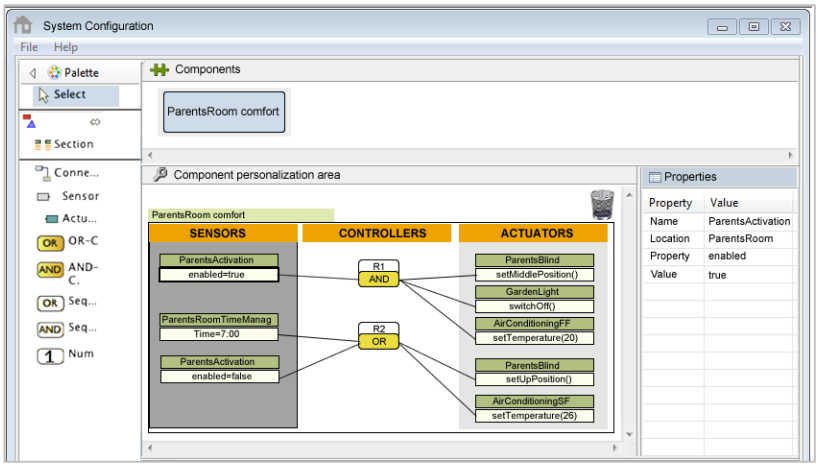


Figure 8.1: Snapshots of PervML and Pantagruel

optimize the creation of the weaving model and transformation rules since they required the most *Modeler expert*'s effort and time.

Finally, we have detected that the *Modeler expert* required more effort to apply Medem whether the role (end-users in this case study) does not have a closer modeling language to be involved in modeling tasks of an existing MDD process (PervML in this case study). In that case, the *Modeler expert* needs to select a proper modeling language for that role, which increases the required effort of the *Modeler expert*. Specially, if the modeling language cannot be reused and it has to be designed from the scratch. Although the selection or design of a proper modeling language could require an important effort of the *Modeler expert*, it could be worth in the execution since a role with different skills is actively involved in modeling tasks of an existing modeling approach, which creates a sense of ownership of the process that makes results more difficult to reject in the future [5].

8.2 UIM - Sketcher Case Study

We applied Medem in two existing MDD processes of the Valencian Regional Ministry of Infrastructure, which are called User Interface Modeler (UIM) [160] and Sketcher [161] [162]. These approaches are focused on the development of web information systems.

UIM supports organization' needs on web information systems (e.g, online procedures that citizens may complete to request subventions, grants, etc.). In particular, UIM allows *software development experts* to specify interfaces and how they are related in an abstract way with the information system. UIM uses concepts in its models such as ClassView, Visualization Set, Filters, Patterns and Package Unit Interaction, so the use of UIM may be complex and it requires that software development experts spend time in learning the modeling language. The upper half of Figure 8.2 shows a snapshot of different UIM models from the UIM editor. Specifically, the figure shows a

subset of UIM models that specify a web interface to allow citizens to manage car licenses. Although UIM use visual elements, its complexity is mainly related to the concepts (which are not familiar for non-software development experts) and the navigability among models (e.g. the *InformationIU* concept is described with a different model that uses distinct concepts such as *visualizationSet* or *classView*).

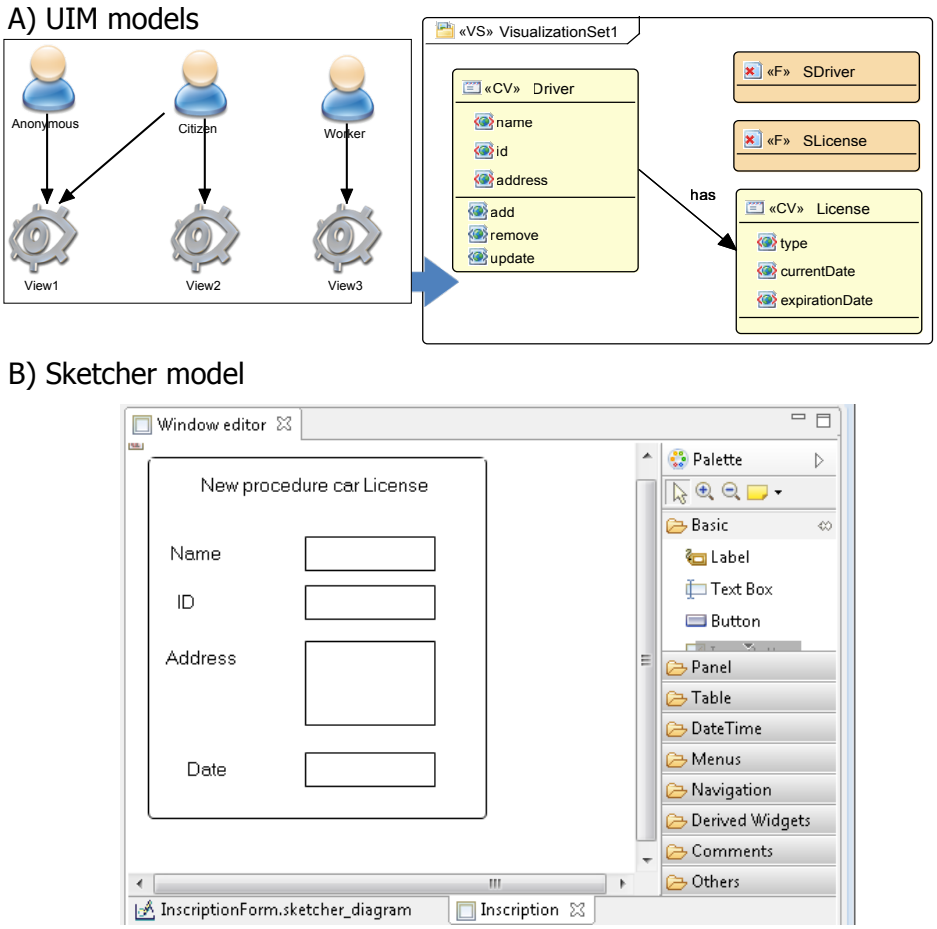


Figure 8.2: Snapshots of UIM and Sketcher

In the organization, *software development experts* use UIM to develop web information systems but they also need to collaborate with *domain experts* in documentation. *Domain experts* in documentation are in charge of designing user interfaces because they know the information that citizens

have to complete to carry out the organization' procedures. However, *domain experts* do not use UIM as *software development experts* do, *domain experts* use Sketcher to express the initial representation of user interfaces in a closer way since Sketcher is based on the standard notation of sketching tools, which favors the users' validation in early stages of development. This situation forces *software development experts* to manage all Sketcher model descriptions in order to translate and integrate them in UIM models. Therefore, we applied Medem in UIM and Sketcher to enable collaborative modeling between *software development* and *domain experts*.

To apply Medem as we previously described, the *Modeler expert* designs the weaving model and transformation rules to bridge all the UIM and Sketcher concepts. The UIM meta-model has about 65 elements and the Sketcher meta-model has about 41 elements. These meta-models are taken as input by the *Modeler expert* to design correspondences in the weaving model. For example, the weaving model bridges the (*ClassView and VisibleAttribute*) UIM concepts and the *Input/Output Widget* Sketcher concepts. Next, the *Modeler expert* extends the UIM editor to support the gap creation and description, respectively. Then, the *software development expert* builds a partially instantiated UIM model with gaps (i.e., the *EditableInformationView* gap to allow domain experts to describe the web form that citizens may complete to renew their car license). These gaps are translated to be completed by domain experts in the Sketcher editor as the snapshot of the lower half of Figure 8.2 shows (i.e., the *EditableInformationView* gap is described to include in the web form some tags and *Input widgets*). Once the *domain expert* completes the descriptions of gaps, the Medem toolkit automatically translates Sketcher model descriptions to UIM descriptions to resolve a fully UIM instantiated model that integrates descriptions of both *software development experts* and

domain experts.

Once Medem was applied in UIM and Sketcher, we consider that Medem provided the main following advantages: (1) *software development experts* were not forced to integrate themselves descriptions from Sketcher models to existing UIM models since the Medem toolkit automatically obtains a fully instantiated UIM model with descriptions from the Sketcher model; and (2) the variability model avoided isolation features in the Sketcher-UIM integration because it keeps the relation with each concern that Sketcher description fulfills and the elements of the UIM model.

At the end, we also detected in this case study that the initialization phase of Medem using the toolkit (designing the weaving model, obtaining the model transformations and extending the model editor) may require more time than the remaining steps of Medem. However, this phase is reusable even though *domain experts* change model descriptions or *software development experts* create new gaps.

Moreover, we detected a problem that appears in the transformation that the toolkit automatically carries out if Sketcher descriptions cause a conflict with UIM descriptions. For example, the same component is already used in UIM descriptions, or the same component is used to different purposes (e.g., the *software development expert* designs a field named ID in UIM to request the user the personal identification number, whereas the *domain experts* adds a ID field in Sketcher to request the user the license plate). This problem triggered the extension of the Medem toolkit to show an information message in the extended M_b editor (the Sketcher editor in this case study) if this conflict occurs after gap descriptions are transformed. To achieve this, we defined a set of rules and we also use EMF Model queries between gap descriptions and the m_a model.

8.3 Bioengineering Kinematic - Medical Protocol Case Study

We applied Medem to overcome the dependence on using new technology between biomedical engineers and doctors of a local hospital [163]. The local hospital provides equipment that is made of cameras, and sensors to track patients' movements. It also provides commercial tools named *Capture, Tracker, and Analyzer* [164] (from now onward Bioengineering Kinematic) to describe and analyze biomechanical protocols. Biomechanical protocols are used to measure performances and identify changes in human body movements and muscles (i.e., a movement produces pain or not in patients). For example, a measure that can be obtained from a movement is the range of motion. It provides the maximum degrees that are achieved in a movement.

To specify this in the tool, the biomedical engineer requires knowledge about (1) how this measure can be obtained, and (2) how to combine and use the operators that the tool provides. In short, many operators are required in the tool to get the range of motion of a movement such as: the interpolation of the points involved, the application of several functions (in order to calculate vectors, angles, midpoints, constants of the points involved), the creation of 1D plains, and events to measure the range of motion by using the functions that are previously created. The upper half of Figure 8.3 illustrates some operators used in the tool to define the biomechanical protocol for shoulder, which can be set to be repeated several times to detect differences. This protocol is focused on analyzing data of four existing shoulder movements: flexion, extension, abduction, and adduction. In particular, the figure shows a subset of the 93 operators and 128 variables that are need to analyze the biomechanical protocol and get a report.

Although Bioengineering Kinematic provides advantages for the patient,

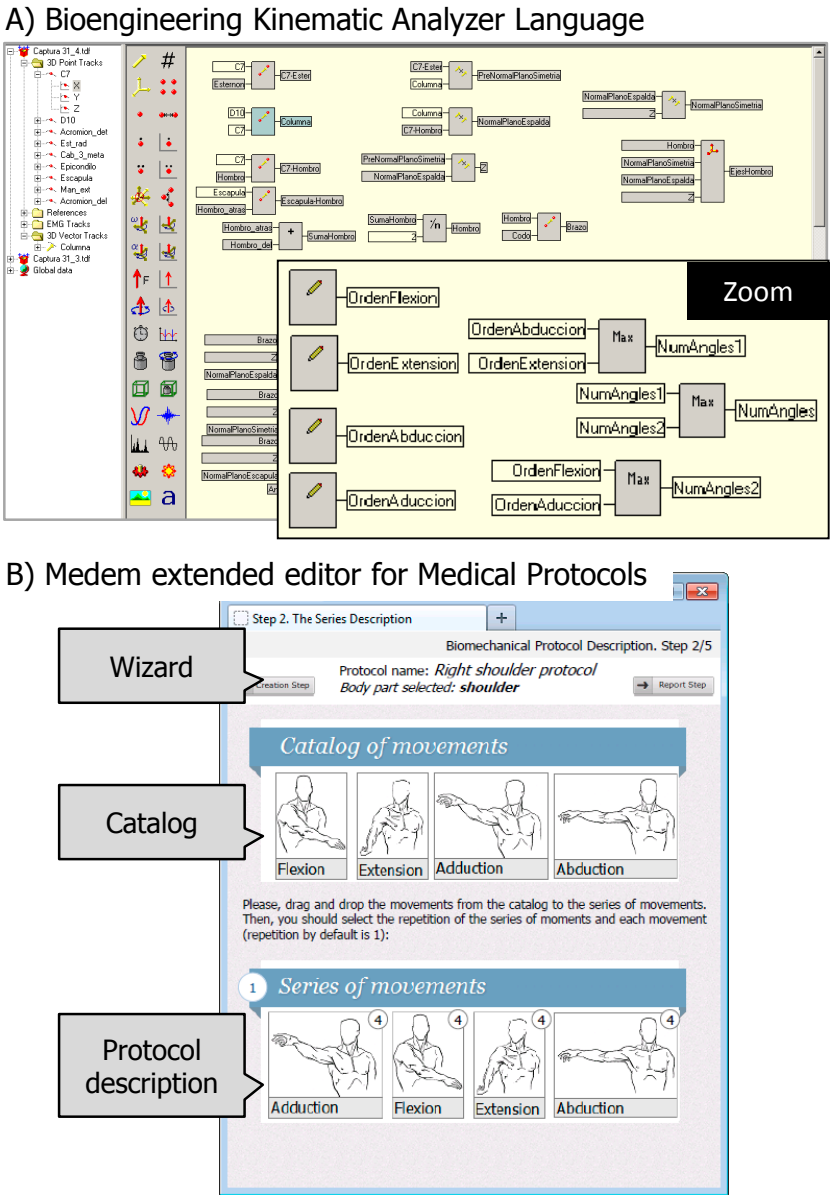


Figure 8.3: Snapshots of Bioengineering Kinematic Analyzer and Medical Protocol

doctors do not use them due to the complexity, so they depend on biomedical engineers. This is because the tools use concepts that doctors are not familiar with (such as projections, vectors, or signals) while doctors use different concepts according to their medical activity (such as series, movements, or

muscles). As a result, biomedical engineers carry out the descriptions of biomechanical protocols according to the doctors' descriptions. This causes a tedious process between doctors and biomedical engineers that could be solved if doctors themselves describe the protocols because doctors are who best know both the protocols they would like to analyze and the information that the report should include to each patient. To tackle this, we apply Medem to enable collaborative modeling between biomechanical engineers and doctors.

To start with, we designed a DSL named Medical Protocol [163] since, to the best of our knowledge, there is no an existing DSL that fits the concepts of biomechanical protocols. Figure 8.4 shows the main concepts of the DSL. The DSL includes: (1) the concepts that doctors are familiar with (see the upper side of the figure), (2) the concepts that support movement and muscle measures (see the dashed blue boxes in the middle side of the figure), and (3) the concepts that allow doctors to design reports according to their preferences (see the dotted red boxes in the bottom side of the figure).

Figure 8.4 also shows how the DSL concepts are related. A brief description of the main concepts is as follows:

- **Protocol.** It is the root concept to measure performances in series of human body movements. It has to be mainly related to a Patient.

- **Series.** It is composed by one or more movements, which can be repeated one or more times.

- **Movement.** It represents a movement that can be performed by patients. At the moment, doctors are focused on analyzing knee and shoulder movements, so we specialize Movement in Knee and Shoulder. Moreover, we specialize each one with the available existing movements. Shoulder is mainly specialized in Flexion, Extension, Abduction, and

Adduction. Knee is mainly specialized in Flexion and Extension. Each movement can be repeated one or more times, and it involves one or more muscles.

- **Muscle.** It expresses the muscles that can be analyzed in a movement.
- **Measure.** It represents measures that can be obtained by analyzing Muscle or Movement data.
- **Movement Measure.** It is related to the movement that analyzes. Moreover, we analyze the data that is usually measured in a movement and we specialize Movement Measure to represent the available measures such as angle, velocity and frequency.
- **Muscle Measure.** It is related to the muscle that analyzes. Moreover, we also analyze the data that is usually measured in a muscle and we specialize Muscle Measure to represent the available measures such as frequency and RMS (*Root Mean Square* used to analyze the fatigue).
- **Report.** It expresses the report that doctors should design to review the output data for the selected measures using plots and tables.
- **Plot.** It represents data about a measure of a muscle or movement.
- **Table.** It represents data about measure of a muscle or movement. A table is composed by one or several rows (each row is represented with the *RowData* concept).

Thus, these concepts provide expressiveness to doctors using concepts they are familiar with rather than force them to learn how to specify each movement and measure using the concepts of the existing *Analyzer* tool.

Regarding to the DSL view, it follows the guidelines and design decisions that were identified in Chapter 4 to involve users in modeling tasks. We

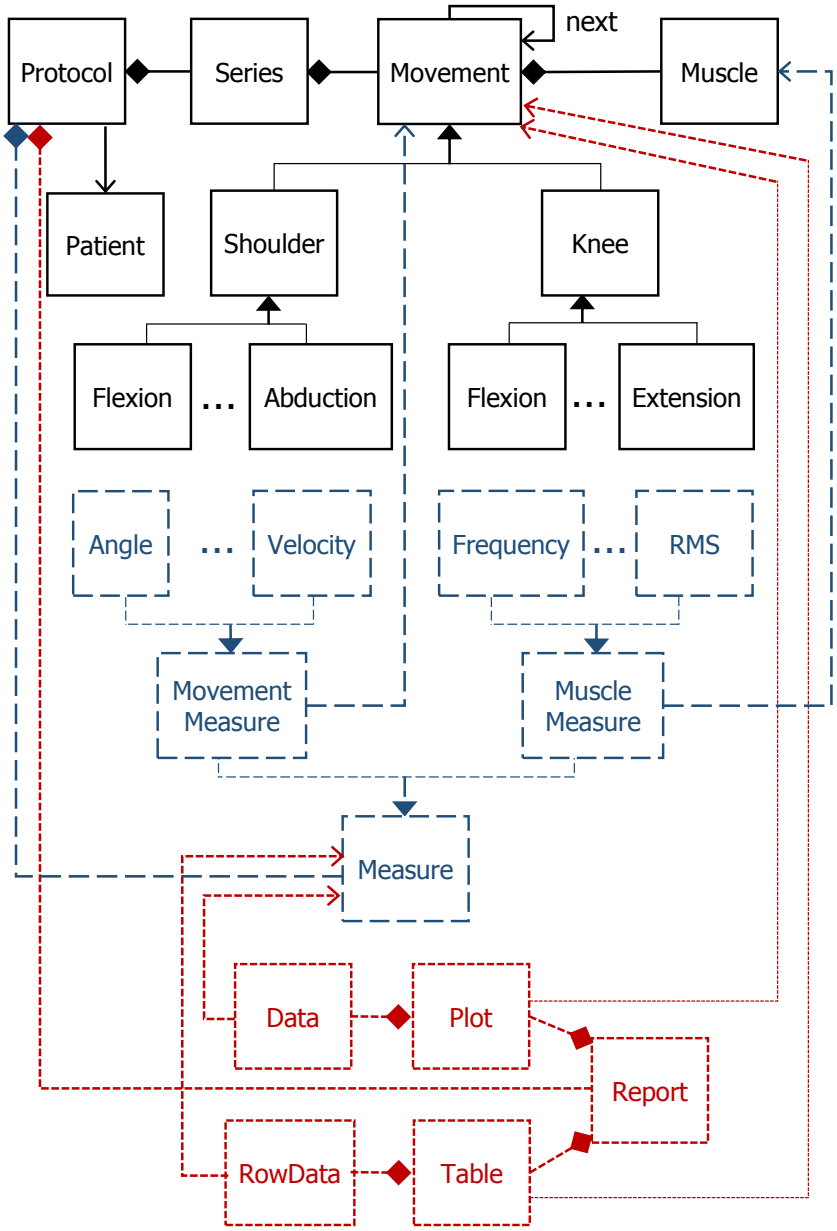


Figure 8.4: The Medical Protocol meta-model

design the DSL view by taking two main design decisions: (1) a catalog of movements and measures, and (2) a wizard. The description of each one and how they are shown is as follows:

A catalog of movements and measures. We decide to design a catalog

of movements and measures to overcome the design barrier that we detected in this case study by following the design suggestion proposed by Ko et al. [165]. In addition, we detect that a catalog perfectly fits in the description of biomechanical protocols because we notice that doctors use (1) the same kind of movements (doctors do not create body movements, they use the existing ones), (2) the same muscles are involved in a specific movement, and (3) the same measures can be analyzed in different protocols.

The view of the catalog is different for Movements and Measures. Each Movement is shown as a box with a representative image, name, and the property to set the number of repetitions (the number of repetitions by default is one). Each Measure is displayed using a tree as [166] recommends in a different view to design the report. The tree has two main nodes: Movements and Muscles, which are used to represent the available measures. Each item of the tree includes the measure name and a brief description.

A wizard. We decide to include a wizard to overcome the selection and use barriers by following the Welie et al. [166] recommendation. The wizard helps doctors to describe a protocol since several decisions need to be made. In the wizard, we design five steps that doctors should follow to describe a protocol. In addition, we use navigation buttons to guide doctors among the steps. The steps of the wizard are the following:

1. **The protocol creation.** It allows doctors to create a new protocol description by setting a name to identify it and selecting the body part to analyze (shoulder or knee). For example, a doctor creates a new protocol named *Right shoulder protocol* and selects shoulder as body part.

2. **The series description.** It allows doctors to select the movements that describes a series of movements. The available movements are shown from the catalog according to the body part selected in the previous step. For example, the series are specified to be repeated once and it is made up of four movements (flexion, extension, abduction, and adduction). Each movement is specified to be repeated four times.

3. **The report definition.** It allows doctors to design the elements (plots and tables) that the report should include. For example, the report of this protocol includes a table. For each row, an angle movement measure is selected for each movement.

4. **The patient selection.** It allows doctors to relate the protocol that has just been described for a patient. Thus, the patient can perform the movements according to doctors' descriptions.

5. **The protocol storage.** It allows doctors to store the protocol. The view of this is just a button. However, we would like to highlight the importance of this step because of it depends that the protocol is stored using the DSL concepts and transformed into concepts of the existing *Analyzer* tool.

We implement an interface prototype according to the DSL view presented above. The lower half of Figure 8.3 shows a snapshot of the interface prototype of *The Series Description* step. Specifically, the figure shows (1) the head and navigation buttons (see the upper side of the snapshot); (2) the catalog of movements for shoulder (see the middle side of the snapshot); (3) the area to describe and set the series of movements and repetitions by using drag and drop from the catalog, and selection for the number of repetitions for each movement (as the snapshot shows, the series of movements is made

up of four movements and each one has set 4 repetitions);

After the Medical Protocol DSL and interface are created, the *Modeler expert* applies Medem by designing the weaving model and transformation rules to bridge all the Bioengineering Kinematic and Medical Protocol concepts (Step 1 and 2). The Bioengineering Kinematic meta-model has about 118 elements among meta-classes, classes and attributes and the Medical Protocol meta-model has about 51 elements. For example, the weaving model bridges the (*Projection, Angle, Vector*, and *Constant*) Bioengineering Kinematic concepts and the *Flexion* Medical Protocol concept.

Next, the *biomedical engineer* builds a partially instantiated Bioengineering Kinematic model (Step 4 of Medem) with gaps (i.e., a gap to allow doctors to describe protocol with a series of patient's movements for shoulder). These gaps are translated (Step 5 of Medem) to be completed by doctors in the Medem extended editor for Medical Protocol (Step 6 of Medem) as the snapshot of the lower half of Figure 8.3 shows (i.e., the movements of the protocol are made up of a series with four different movements: abduction, flexion, extension, and abduction). Note that the snapshot shows the view that allows doctors to set the movements of the protocol. Further information about the DSL view and the remaining steps of the wizard can be found in [163]. Finally, the Medem toolkit automatically translates Medical Protocol model descriptions to Bioengineering Kinematic descriptions (Step 7 of Medem) to resolve a fully instantiated Bioengineering Kinematic model.

As a result, the application of Medem in this case study required more *Modeler expert's* effort than the other case studies (i.e., it was also necessary to built both the Medical Protocol DSL and its editor). Nevertheless, the application of Medem overcomes the barriers that doctors had with the Bioengineering Kinematic tool, so the *Modeler Expert* effort in the specification is worth in the execution by providing the following main

advantages: (1) doctors are able to carry out themselves descriptions of biomechanical protocols, (2) doctors manage less concepts, and (3) movements and measures are reused for each protocol rather than force doctors to describe them from scratch.

8.4 Conclusions

In this chapter, we conclude by discussing the results of the validation of this thesis proposal (Medem) in the three case studies that have been presented.

Overall, the application of Medem in existing modeling approaches reveals positive results regarding its applicability and feasibility. We therefore believe that the application of Medem in different domains provides representative results to indicate that it could be applied for involving users in modeling tasks of other MDD processes. Next, we discuss the advantages of Medem, its drawbacks and future improvements.

With regard to the advantages, Medem actively involves different types of users in modeling tasks of an existing MDD process using a closer modeling language for them, which helps those users in the description of domain specific content [7] by do not force them to use different modeling primitives to actively participate in a modeling project [8]. Furthermore, Medem integrates users' descriptions to obtain a unified system description. Thus, Medem lowers the barrier of users in MDD approaches, which promotes the universally adoption of MDD [1]. Moreover, Medem provides collaborative modeling mechanisms to delimit the concerns of the system in which users may participate, which guides users to be focused on the concerns of the system that they may collaborate rather than be focused on the description of the full system [5, 13]. In addition, Medem exchanges model descriptions between different modeling languages using interoperability mechanisms that

are non-intrusive with the structure of such models, which promotes the application of the approach in existing MDD processes. All in all, the users, who were involved in the case studies, expressed by means of interviews and thinking aloud the usefulness of the approach since Medem enables them to actively collaborate in the description of concerns by using concepts that they are familiar with. Even though the users expressed that it would be useful for them that the tool provides them feedback using information messages (e.g., users might like to know whether there are some conflicts or errors in the system due to their descriptions), users expressed their satisfaction with Medem due to the guidance throughout the description of concerns since the tool only requested to describe those concerns of the system that rely on them.

With regard to the drawbacks, we detect that the specification phase of Medem requires the most-time consuming (e.g., to specify the correspondences in the weaving model and transformation rules) as the three different applications of our approach have revealed. This result fits in the literature since the specification of the weaving model and transformation rules is a challenging task that demands the most *Modeler Expert*'s effort and time [39]. This required time is aggravated by the fact that meta-models may become very large: for instance, the UML 2 meta-model [47] has about 260 meta-model classes [167]. Even there are approaches such as [39, 75, 76] that seek the improvement of modelers' productivity by reducing the specification time of model transformations by means of semi-automatic approaches and generic model transformations, model transformations are mostly created manually and adhoc [76].

Hence, some aspects of the tool support have to be improved in our approach to reduce the effort that is necessary to initialize our approach. As the next chapter presents, we started working on providing mechanisms

to only define the correspondences in the weaving model that support the description of gaps rather that the fullset of concepts since its description may not involve as many concepts as the modeling language has. Thus, the number of correspondences can be reduced as well as the necessary effort of the *Modeler expert*.

As additional improvement, a major issue is to reduce the time that is required to create the transformation rules. Our goal is to reduce the number of necessary correspondences in the weaving model and therefore, the number of model transformations will be also reduced. Another goal is to provide generic model-to-model transformation rules that work as easy as a language translator by only selecting the weaving model and the source model as input to create a target model as output. Thus, the *Modeler expert* does not have to provide the transformation rules and they will be also reusable in different modeling approaches.

Another future improvement is the model validation mechanisms because we detected some problems during the case studies (e.g., user-dependent descriptions do not fit the boundaries of the variability model). Despite we address a great number of problems integrating model validation mechanisms in the toolkit by means of a set of domain-independent rules using EMFMQ, we believe that it is important to provide more validation mechanisms for checking syntactic and semantic problems in models (e.g., user-dependent properties may cause conflicts with the properties of the common system description).

Chapter 9

TOWARDS THE EFFICIENT SPECIFICATION OF THE INTEROPERABILITY MECHANISMS

The specification of mechanisms to support collaborative modeling from different modeling languages (the weaving model and transformation rules) demands the most *Modeler Expert*'s time and effort as concluded in Chapter 8 since the support of the full set of meta-model concepts is mostly manual [76] and performed at the beginning of the process even though there are approaches that achieve a semi-automatic specification of model transformations. These approaches reduce the development time [39] but they could also require manual refinements (since the complete automation is too ambitious except from simple transformations [39]).

Therefore, this chapter presents our ongoing work for achieving an efficient support of collaborative modeling from different modeling languages by extending Medem since we detected in the case studies that some correspondences of the weaving model and their corresponding transformation rules are not used throughout collaborative modeling. This is because the

necessary concepts to fulfill the gaps do not involve as many concepts as the base modeling language has and therefore, it is not necessary that the full set of both correspondences and transformation rules is specified at the beginning of the specification phase.

Hence, we propose a strategy, named Medem-on-demand, which only includes on-demand correspondences in the weaving model that are necessary to support collaborative modeling. We join the term "on-demand" to stress the fact that both the weaving model and transformation rules only include concepts that support collaborative modeling (the descriptions of gaps) rather than create the full set at the beginning. Therefore, Medem-on-demand can reduce the number of correspondences in the weaving model, the transformation rules, and the *Modeler Expert*'s effort.

The rest of this chapter is organized as follows: Section 9.1 presents some existing approaches that accelerate the specification phase using model transformations by-example and their drawbacks. Section 9.2 describes Medem-on-demand. Section 9.3 presents the extension of the Medem toolkit to support Medem-on-demand. Section 9.4 describes the application of Medem-on-demand in the same case studies that Medem was applied for evaluating its applicability, its feasibility and comparing the results with Medem in order to assess whether Medem-on-demand increases *Modeler experts'* productivity by reducing the number of model transformations. Finally, Section 9.5 concludes the chapter.

9.1 Model Transformations By-Example

Model transformations can be used in many different application scenarios, for instance, to provide interoperability from different domains, from different roles, and from different software representations as explained in Chapter 2.

However, model transformations are typically developed manually and they are becoming more and more complex [76, 168].

As a consequence, research groups start working in approaches for easing the burden of specifying model transformation rules by hand such as Model Transformations By-Example approaches (MTBE) [168]. MTBE approaches [169, 170] follow the same idea as querying database systems by giving examples of query results and programming by-example for demonstrating actions, which are recorded as replayable macros [32].

In general, two kinds of approaches may be distinguished in MTBE [171]: (1) approaches following a demonstration-based approach in which model transformations are shown in the model editor by modifying example models. These modifications are recorded and can be applied to other models as well; and (2) approaches that follow a correspondence-based approach in which the input model, the output model as well as the correspondences between them have to be given by the user. For both kinds, a multitude of approaches have been proposed during the last years such as [168, 171, 170, 172, 173, 169]. Next, we introduce as example some MTBE approaches and other ones that seek to ease the specification of model transformations.

On the one hand, Langer et al. [168] and Kappel et al. [171] seek to accelerate the model transformation specification phase by defining model-to-model transformations by-example. These approaches take as input to build model transformations as a triple comprising a source model and a target model that are taken as example, and a weaving model with correspondences between these two models.

On the other hand, Bollati et al. [39] provide mechanisms to semi-automate the development of transformations. Moreover, there are approaches such as the proposed by Cuadrado et al. [75] that brings generic model transformations to promote reusability. These approaches can acceler-

ate the development time of transformations, or it can favor the reuse of model transformations if the modeling languages change. In addition, Didonet et al. [76] propose the automatically production of the weaving model using matching transformations and two input meta-models.

Although these approaches reduce the necessary *Modeler experts*' effort to develop model transformations and they aim for semi-automated transformation generation, the generated transformations are intended to be further refined by the user [171, 39] since complete automation is too ambitious, except from very simplistic transformations [39]. Moreover, these approaches include the full set of correspondences in the weaving model and model transformation at the beginning. However, these approaches are focused on translating an entire source model to an entire target one rather than complete a partially instantiated model with model fragments of a different modeling language to support collaborative modeling.

Since we have detected in Medem that some correspondences are not used throughout collaborative modeling even they are defined at the beginning in the weaving model and their corresponding model transformations, the full set of both correspondences in the weaving model and transformation rules is not necessary to included it at the beginning. This is because the necessary concepts to fulfill the gaps with model fragments of a different modeling language could not involve as many concepts as the modeling language has.

Therefore, we propose a strategy towards an efficient support that reduces both the number of correspondences in the weaving model and transformation rules by creating them on-demand rather than specifying the full set at the beginning.

9.2 Medem-on-demand

Although Medem enables collaborative modeling from two different modeling languages to obtain a model that integrates descriptions of both, it requires that a *Modeler expert* designs the fullset of model transformations in the specification phase. As previously explained, the *Modeler expert* designs the model transformations by specifying correspondences among meta-models to bridge every concept of both the Meta-Model$_a$ and the Meta-Model$_b$ in the weaving model.

According to our previous experiences of applying Medem in different domains [152, 162, 163] that were presented in Chapter 8, the specification of the weaving model and the model transformation rules requires the most *Modeler Expert*'s effort (specially if the meta-models have a high number of concepts). Moreover, we observed a phenomenon in our previous experiences that shows that some correspondences of the weaving model are not used (and therefore, some transformations rules neither) since collaborative modeling descriptions do not involve as many concepts as the language has.

For exemplifying the phenomenon that we observed, we use the CD-ER example that was presented in Figure 6.5 of Section 6.2. At this point, it is important to highlight that the model transformations were created at the beginning of Medem by a *Model expert*, so the model transformations cover the fullset of CD and ER concepts, which requires the most *Modeler expert*'s effort. This required effort is aggravated by the fact that meta-models may become very large: the CD meta-model [47] has about 106 elements among meta-model classes and attributes. The ER meta-model [174] has about 52 elements.

To support Medem in this example, around 72 element relationships need be added in the weaving model at the beginning. We focus on collecting

the number of the necessary correspondences of the weaving model in order to compare the *Modeler Expert*'s effort because these correspondences are used to create the transformation rules. According to [115], this comparison should give an indication on how much effort requires the creation of both the weaving model and the transformation rules. Although there are different proposals of CD and ER meta-models, they may require a similar order of magnitude to specify the full set of correspondences and transformation rules.

However, note that the CD-ER example does not use all the CD or ER concepts in the creation of description of gaps such as inheritance or associative entity. Therefore, some correspondences of the weaving model and some transformations rules are not being used. In this example, it is only necessary to add 33 element relationships in the weaving model to the fully instantiated CD model shown in the bottom part of Figure 6.5. As a result, the number of correspondences in the weaving model is reduced the 45.8% with regard to create the full set of model transformations at the beginning.

Although, the CD-ER example is used to illustrate both collaborative modeling from different modeling languages and the phenomenon that shows that some correspondences of the weaving model are not used for supporting collaborative modeling descriptions, the numbers that are shown in the evaluation section reveal that this phenomenon frequently occurs in different application domains.

Therefore, it is necessary an approach towards an efficient support of collaborative modeling from different modeling languages for application domains in which meta-models may become large, collaborative model descriptions do not involve as many concepts as the language has, and model transformations have to be created from scratch. To address this, our strategy follows the idea of including on-demand correspondences in the weaving model

rather than the fullset of concepts.

The more complex the correspondence requirements in the weaving model are, the more time-consuming specifications will be [21]. To achieve a constant project progress, Klar et al. [21] suggest an iterative approach to define the correspondences. Thus, we propose that the definition of the correspondences in the weaving model is incrementally completed step by step on-demand according to the concepts that are necessary to support collaborative modeling (the description of gaps using a different modeling language). This is advantageous, compared to producing all the correspondences in the weaving model (and the corresponding transformation rules) in one single step [21].

To achieve this, we propose the Medem-on-demand method. Medem-on-demand acts as an extension of Medem for accelerating the specification phase. In this phase, the *Modeler Expert* creates on-demand correspondences in the weaving model. In particular, we propose that the *Modeler Expert* takes as input a source model with gaps to only include in the weaving model those concepts and correspondences that are needed to describe the gaps using a different modeling language. As the *Modeler Expert* does not include concepts that are not going to be used in gap descriptions, the number of correspondences in the weaving model can be reduced and therefore, the number of transformation rules can be also reduced.

Medem-on-demand is illustrated in Figure 9.1. Note that the shaded artifacts in the figure serve to show the changes in Medem-on-demand with regard to Medem.

Figure 9.1 also shows the steps of Medem-on-demand corresponding to the different numbers shown, involved artifacts and roles in its specification and execution phases. These steps are described in detail below:

Step 1. The *Modeler Expert* should extend the model editors for supporting the creation and description of gaps as previously described in Step 3

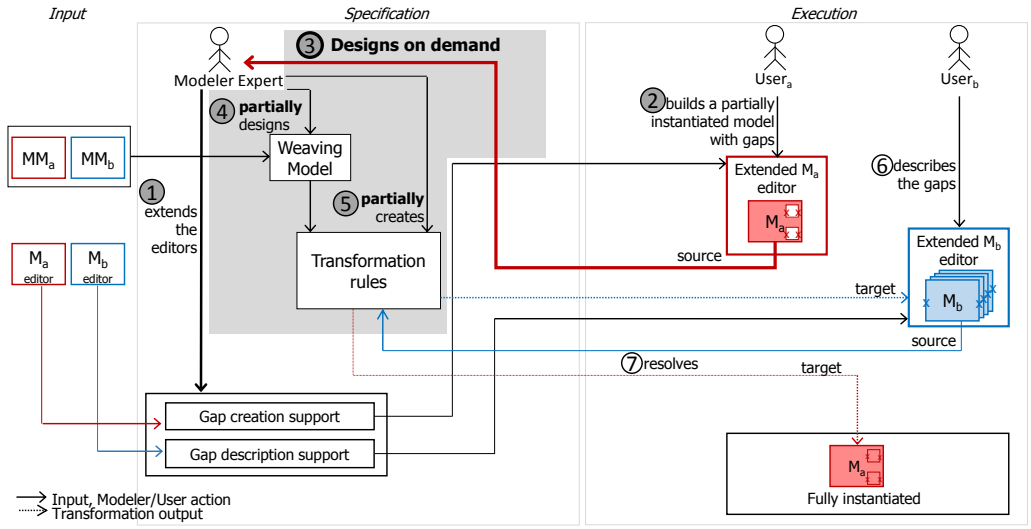


Figure 9.1: Steps of Medem-on-demand during its specification and execution

of Medem.

Step 2. The User$_a$ reuses or builds a partially instantiated model$_a$ and creates gaps using the extended M$_a$ editor as previously described in Step 4 of Medem.

Step 3. The *Modeler Expert* takes as input the partially instantiated model$_a$ with gaps for creating a weaving model on-demand. This weaving model may only include the necessary Meta-Model$_a$ (MM$_a$) concepts and correspondences to support the description of gaps using Meta-Model$_a$ (MM$_b$) concepts.

To help the *Modeler Expert* with the creation on-demand of the weaving model, we design a set of generic model queries that automatically obtain a view of the MM$_a$ concepts that the *Modeler Expert* should include in the weaving model. Thus, the *Modeler Expert* does not have to manually determine which MM$_a$ concepts are necessary. This is

specially useful in order to prevent that necessary MM_a concepts have not been included in the weaving model.

The set of queries takes as input the partially instantiated $model_a$ with gaps (which are stored in the variability model) in order to get the following:

1. The gaps that are stored in the variability model. For example, the gap for refining the sales of the CD-ER example.
2. The MM_a concepts that have been involved in the description of each gap. By following the motivating example, the queries obtain CD concepts for the gap such as: *Class, Association and Property.*
3. The MM_a concepts that are set as boundaries for each gap. Following the example, the gap of the motivating example has the *Class* CD concept as boundary.

Once a query that gets MM_a concepts is executed, each MM_a concept is stored (whether it has been not included yet). Thus, the *Modeler Expert* is provided with the view of the MM_a concepts that may be included in the weaving model.

Step 4. The *Modeler Expert* uses the view of MM_a concepts in order to partially design a weaving model with correspondences for these MM_a concepts with MM_b concepts. Note that we use partially to stress that this weaving model does not bridge all the MM_a concepts, it is focused on including the MM_a concepts that are needed to support the description of gaps.

By following the motivating example, the *Modeler Expert* checks the view of the CD meta-model concepts (such as *class, association, name, type*, and *property*), and includes correspondences for them in the

weaving model. Thus, the number of elements of the weaving model has been reduced.

Step 5. Once the weaving model is created, the *Modeler Expert* creates the transformation rules to transform a source model (i.e., the CD model in the motivating example) conforming to its input meta-model into a target model (i.e., the ER model in the motivating example) conforming to its meta-model as described in Step 2 of Medem. Note that the difference is the reduction of the necessary transformation rules due to the reduction of concepts and correspondences in the weaving model.

The remaining steps of Medem-on-demand (Step 6 and 7) are as the ones described in Medem for describing the gaps and obtaining the fully instantiated model$_a$.

At this point, it is important to highlight that although Medem reuses the weaving model and transformation rules even the MM$_a$ and MM$_b$ concepts involved in the creation and description of gaps change, Medem-on-demand is iterative in that case. Although Medem-on-demand could accelerate the model transformation specification phase since the correspondences in the weaving model are adjusted to the MM concepts that support the description of gaps, Medem-on-demand could require that the *Modeler Expert* adjusts the weaving model and transformation rules if the MM concepts that are involved in the creation and description of gaps change in successive collaborative descriptions.

9.3 Tool Support

In this section we describe the extension of the Medem toolkit to support Medem-on-demand. On the one hand, we define a set of rules that are

executed to select between Medem or Medem-on-demand according to the provided inputs. Medem-on-demand accelerates the specification phase since the *Modeler expert* only has to extend the M_a editor to enable the $User_a$ to build a partially instantiated $model_a$ as described in Step 2.

Only if the partially instantiated $model_a$ with gaps (the CVL variability model) is provided in the specification phase, the rules will select the Medem-on-demand support. By contrast, if both the weaving model and transformation rules of the full set of meta-model concepts are provided, the rules will select the Medem support.

On the other hand, the support of the model transformations on-demand comprises a set of generic model queries that provides the *Modeler expert* with a list that suggests the MM_a concepts that should be included in the weaving model for supporting the description of gaps. Thus, the *Modeler Expert* does not have to manually determine which MM_a concepts are necessary, which is specially helpful when the number of meta-model concepts grows. Also, the list prevents that the *Modeler Expert* forgets the inclusion of some necessary MM_a concepts in the weaving model.

We use the EMFMQ framework as the main building block to create select statement queries for obtaining the list. The queries takes as input the partially instantiated $model_a$ with gaps (which are stored in the CVL variability model) in order to obtain the following (as described in Step 3 of Medem-on-demand): (1) the gaps that are stored in the variability model, (2) the MM_a concepts that have been involved in the description of each gap; and (3) the MM_a concepts that are set as boundaries for each gap.

For example, we have implemented the following select statement query using EMFMQ that selects all the concepts that have been set as boundaries for each gap:

```
1 SELECT statementBoundaries =
2  new SELECT(
3    new FROM(resource.getContents()),
4    new WHERE(
5      new EObjectTypeRelationCondition(
6            CvlPackage.Literals.PLACEMENT_BOUNDARY_ELEMENT
                 ,
7            TypeRelation.SAMETYPE_OR_SUBTYPE_LITERAL)));
```

Following the CD-ER example this query returns the *Class* CD model concept since it is the one set as boundary. Once the set of queries is executed, the list of meta-model concepts is provided and taken as input by the *Modeler expert* in order to only bridge those meta-model concepts in a weaving model.

Figure 9.2 shows a snapshot of the Medem-on-demand toolkit once the CVL variability model that supports the watch example introduced above is taken as input and it is executed the operation that gets the list with all necessary CD model concepts (MM elements) to support the description of gaps.

As figure shows, the Medem-on-demand toolkit recommends that the weaving model should include correspondences for the concepts that are necessary to to refine the gap of how the sales are stored in the information system. Some of the concepts found are: *class*, *association*, *type*, and *property* CD model concepts.

To conclude, we would like to highlight that the toolkit supports Medem as well as Medem-on-demand and it automatically selects between them according to the *Modeler expert* inputs. In addition, the toolkit does not modify the meta-models of the modeling languages and it is customizable to different modeling languages. Moreover, the Medem-on-demand toolkit includes generic queries to get list with the suggested the MM_a concepts, so

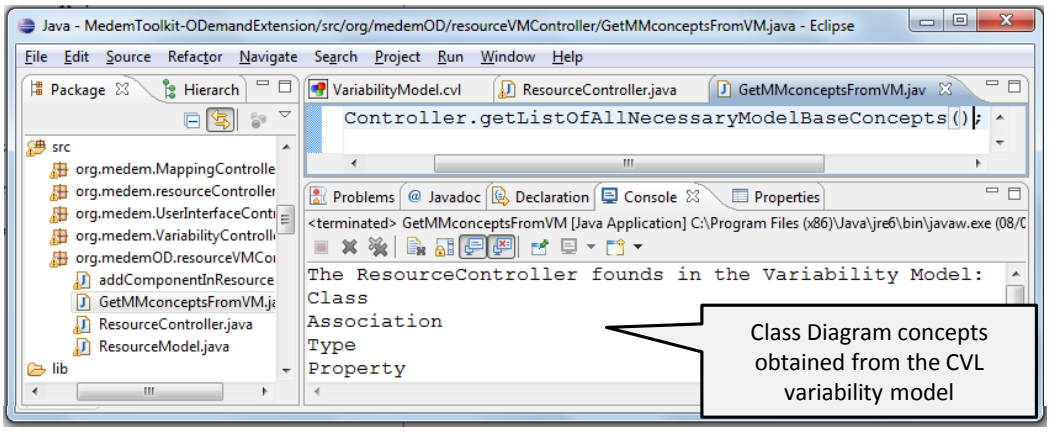


Figure 9.2: Snapshot of the Medem-on-demand toolkit

they can be reused even though the modeling languages change.

9.4 Application and discussion

To assess whether Medem-on-demand increases Modelers' expert productivity by reducing the number of model transformations, we apply it in the same case studies that Medem was applied in order to evaluate its applicability, feasibility and compare the results with Medem.

In particular, we describe for each case study the following: 1) the number of correspondences that were necessary to apply Medem, 2) the total number of model of correspondences that have been set and non-used in Medem, and 3) the application of Medem-on-demand and the total number of correspondences created.

1. **PervML - Pantagruel Case Study**. According to the data summarized in Table 9.1 (see first row), 62 correspondences are needed between PervML and Pantagruel in the weaving model.

   After, we analyze the PervML and Pantagruel model descriptions

and model transformations and we collect that 24 correspondences of the weaving model are not used. This is mainly because the Pantagruel sensor-controller-actuator development paradigm can be corresponded to several concepts of three different PervML models which follows Event Condition Action (ECA) rules. In particular, these PervML concepts are focused on describing not only how the devices and operations works but also how different services are orchestrated. In fact, end-user's descriptions in Pantagruel seek to only orchestrate different services in order to fit them with end-users' preferences. For this reason, we apply Medem-on-demand to focus the correspondences of the weaving model on supporting descriptions of services by orchestrating services and changing their attributes.

By applying Medem-on-demand, the *Modeler expert* extends the editors (Step 1) and executes our approach to take as input both the partially instantiated model with gaps (Step 2) and the list of the necessary meta-model concepts (Step 3) in order to partially design the weaving model (Step 4) and transformation rules (Step 5). We analyze the weaving model and we collect that the *Modeler expert* created 36 correspondences rather than the 62 that were created in Medem.

Therefore, Medem-on-demand reduces the 41,94% the number of necessary correspondences in the weaving model with regard to Medem (see last column of Table 9.1). Moreover, we analyze different PervML smart home descriptions once Medem-on-demand is applied and we notice that the gaps created in the partially instantiated PervML model by the *PervML analyst* comprise the same concepts, so Medem-on-demand does not require new *Modeler expert*'s iterations to neither create new correspondences in the weaving model nor transformation rules.

2. **UIM - Sketcher Case Study**. According to the data summarized in Table 9.1 (see second row), 86 correspondences are needed between UIM and sketcher in the weaving model.

   Then, we analyze the UIM and Sketcher model descriptions and model transformations and we collect that 18 correspondences of the weaving model are not used. This is because the *software development expert* created gaps to allow domain experts to design web interfaces with the data that may be collected from citizens rather than design web interfaces to show information. This makes that some Sketcher concepts are not used even though their correspondences are included in the weaving model. For example, the *TabularPanel* UIM concept was not used during descriptions because it shows information in a table format.

   By applying Medem-on-demand, we analyze the resulting weaving model and we collect that the *Modeler expert* created 61 correspondences rather than the 81 that were created in Medem. Therefore, Medem-on-demand reduces the 29,07% the number of necessary correspondences in the weaving model with regard to Medem (see last column of Table 9.1).

   Whether the *software development expert* creates gaps in further scenarios that imply the use of concepts that are not included in the weaving model such as the ones related to show information on web interfaces, Medem-on-demand may iterate (only Step 4 and 5) to update the weaving model and the transformation rules. In this case, although the number of correspondences will be the same in Medem and Medem-on-demand, Medem-on-demand still provides three main advantages: an initial reduction of correspondences in the weaving model, reduces the initial effort of the *Modeler expert*, and enables collaborative modeling between the *software development expert* and

the *domain expert* before Medem enables it.

3. **Bioengineering Kinematic - Medical Protocol Case Study**. According to the data summarized in Table 9.1 (see third row), 347 correspondences are needed between Bioengineering Kinematic and Medical Protocol in the weaving model.

   Then, we analyze the model descriptions and model transformations and we collect that 186 correspondences of the weaving model are not used. This is because the gaps are focused on only describing biomechanical protocols for shoulder and the weaving model also includes correspondences for knee movements. Moreover, we notice that doctors have only used tables to design the reports (they do not include plots in gap descriptions).

   By applying Medem-on-demand, the *Modeler expert* takes as input both the partially instantiated model with gaps and the list of the necessary meta-model concepts in order to partially design the weaving model and transformation rules. We examine the weaving model and we collect that the *Modeler expert* created 164 correspondences rather than the 347 that were created in Medem. In conclusion, Medem-on-demand reduces the 52.74% the number of necessary correspondences in the weaving model with regard to Medem (see last column of Table 9.1).

   Whether doctors use plots in future descriptions, a new iteration of Medem-on-demand is carried out to allow the *Modeler expert* to update the weaving model and transformation rules. Moreover, new iterations of Medem-on-demand can be carried out to support the description of biomechanical protocols for different body parts such as knee. We would like to emphasize that in case that the creation or description of gaps comprises new concepts, Medem-on-demand can iterate to

	Medem		Medem-on-demand	
Case study	Total	Non-used	Total	**Result (%)**
1) PervML - Pantagruel	62	24	36	-41.94%
2) UIM - Sketcher	86	18	61	-29.07%
3) Bioengineering Kinematic Analyzer- Medical protocol	347	186	164	-52.74%

Table 9.1: Summary of total and non-used correspondences in the weaving model using Medem and Medem-on-demand in the three case studies

support them. Therefore, Medem-on-demand also provides advantages in this case study from the beginning (such as the reduction of the number of correspondences in the weaving model and the initial effort of the *Modeler expert*, and enables collaborative modeling before Medem enables it) as we detected in the previous case study.

Once Medem-on-demand has been applied to the three case studies, we believe that the data that is summarized in Table 9.1 provides representative results since the three case studies scope different domains and levels of complexity. Furthermore, the case studies conducted have served to show not only improvements of Medem-on-demand with regard to Medem but also some drawbacks.

With regard to the improvements, Medem-on-demand reduces the necessary correspondences in the weaving model to include only those ones that are needed to support gap descriptions rather than include correspondences to bridge all concepts of the modeling languages. Thus, the *Modeler expert* accelerates the specification phase and the execution phase (model

descriptions) can be performed before. In addition, the *Modeler expert* is helped in the design of the weaving model by taking input examples (in particular, the partially instantiated model, the list of the created gaps and the list with the recommended meta-model concepts that may be included in the weaving model), which according to [17] examples help to organize better the system needs and domain reference components.

With regard to the drawbacks, we detect that Medem-on-demand may not always reduce the necessary correspondences in the weaving model, so Medem may be applied instead. We analyze that the reduction of the correspondences in Medem-on-demand depends on:

1. The number of necessary concepts to describe the gaps with regard to the total meta-model elements. For example, as more concepts are demanded to describe the gaps more correspondences are needed in the weaving model, so the weaving model on-demand may have the same number and the specification phase can require more time for the *Modeler expert* to update the weaving model and transformation rules on-demand.

2. The number of changes in the concepts that are demanded to support future gap creation and description. For example, the Bioengineering kinematic-Medical protocol case study shows the best results of the Medem-on-demand application as we described above. However, doctors can demand new elements in gap descriptions in the successive collaborative descriptions such as knee protocols. In this case, a new iteration of Medem-on-demand will be required to allow to *Modeler expert* to change the weaving model and transformation rules.

9.5 Conclusions

In this chapter, we have presented our ongoing work, Medem-on-demand, as an extension of Medem to efficiently obtain model transformations in the specification phase. In order to address this, we propose mechanisms to only specify correspondences in the weaving model and transformation rules that are necessary for describing the gaps rather than create a full set of correspondences and transformation rules at the beginning.

Therefore, Medem-on-demand could accelerate the specification phase since it reduces the number of correspondences and transformation rules. We believe that if the required effort of the specification phase is reduced specifying model transformations on-demand, it will favor the collaborative modeling from different modeling languages.

In order to put Medem-on-demand into practice, we have described the technological decisions to support it in a transparent way to the users once the Modeler Expert ends the specification phase. At this point, it is also worth pointing out that although the example used in this chapter to present Medem-on-demand is focused on CD models and ER models, the need of model transformations on-demand can be transferred to other languages, as we shown in the case studies, in which meta-models may become large, collaborative model descriptions do not involve as many concepts as the language has, and model transformations have to be created from scratch.

Figure 9.3 shows our recommendation, based on the improvements and drawbacks previously discussed, on which method (Medem or Medem-on-demand) should be chosen by *Modeler experts* to achieve an efficient support of collaborative modeling from different modeling languages. For example, if the full set of model transformations is available, or collaborative modeling involves many concepts and the involved ones could change in successive gap

descriptions, we recommend to choose Medem since the weaving model may not be reduced and Medem-on-demand will require different iterations due to the changes in successive gap descriptions. As figure shows, we recommend to choose Medem-on-demand in other cases.

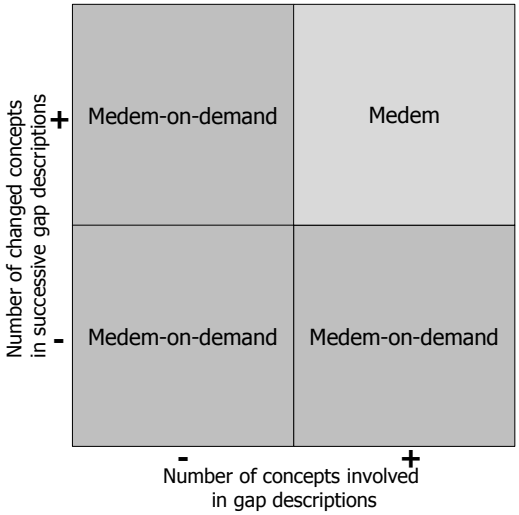


Figure 9.3: Choosing between Medem and Medem-on-demand

In conclusion, we have shown that both our approach and its extension are capable of achieving interoperability between models from different approaches, and providing collaborative modeling mechanisms in a non-intrusive way. Therefore, different types of users are actively involved in modeling tasks according to the confronted thesis goals.

Chapter 10

CONCLUSIONS AND FUTURE WORK

The present work has introduced a model-driven and variability-based approach for confronting the challenge of achieving interoperability from different modeling languages in order to involve users in modeling tasks. Confronting this challenge, this work also enables collaborative modeling in order to delimit and guide users in the modeling effort in a non-intrusive way (i.e., without affecting the structure of modeling languages).

We applied our approach in three different domains in which different types of users (end-users and software professionals; domain experts and software development experts; and doctors and biomedical engineers) are involved in a common modeling project using different modeling languages to perform model descriptions. Whether in smart homes, web information systems or biomechanical protocols, the growing number of modeling approaches underlines the rising relevance of giving different users the opportunity to contribute in model descriptions. We consider that our approach can also be applied to other domains with similar results.

This last chapter reviews our central results and primary contributions, and proposes new areas for future research in connection with the limitations

of this work. First, Section 10.1 presents the main contributions of our approach. Section 10.2 outlines the assessment and future work that can complement and extend this thesis. Section 10.3 provides an overview of the publications that have emerged from this work. Section 10.4 presents the projects directed related to some parts of this thesis. Finally, Section 10.5 concludes with some final remarks.

10.1 Contributions

The major contribution of this thesis is **a model-driven and variability-based approach** for achieving collaborative modeling and non-intrusive interoperability between models of heterogeneous modeling languages to involve users in modeling tasks. This approach combines the main ideas of the End-user Development, the Model-Driven Development and the Variability Management fields to achieve not only interoperability in a non-intrusive way with the structure of models but also, to use variability management in a novel way to enable collaborative modeling from a different modeling language. This main contribution is complemented with three other contributions:

1. The **identification of different user skills and guidelines** for involving users in software development activities, and **their application** for involving end-users in modeling tasks of an existing MDD process for developing pervasive systems by selecting and customizing system features.

2. **A model-based and variability-based implementation** to support the approach in a transparent way to the users once it is initialized.

3. An **extension of our approach** for addressing an efficient support of collaborative modeling by specifying on-demand the interoperability

mechanisms. Thus, this extension reduces the necessary effort.

Although the above contributions push towards the involvement of users in a unified modeling project using different modeling approaches, we believe that this thesis also provides remarkable results for the communities of the combined fields as follows.

- Relevant results for the End-User Development community:
  - **Identification of guidelines** that can be applied in different domains to involve users in modeling tasks (i.e., users should be provided with a closer modeling language and users should be focused on describing user-dependent properties).
  - **Case studies** that are representative of real problems of existing modeling approaches in which users cannot participate even though they are the ones who best know the expected functionality. This is because users need skills to capture every important aspect of their software system through models.
  - **Application of the identified guidelines and interface design decisions** in case studies to involve users in modeling tasks since it was not available a closer language and tools that fit into the some case study goals and identified users.
- Relevant results for the Model-Driven Development community:
  - **Identification of challenges, lessons learned, and design decisions** in collaborative modeling to involve users in modeling tasks (i.e., using variability management to provide users with guidance throughout the modeling process).
  - **Empirical evidences of integrating submodels or models descriptions** that are made from different participants who ac-

tively participate in the modeling effort using the CVL variability management language. CVL identifies variation points in a base model and they may have models to describe them. In addition, we propose that these models are described using a different modeling language and we provide mechanisms to translate and integrate them in the base model. This could help to achieve a wider adoption of MDD processes.

  - **Resolution of conflicts** during the integration of such submodels or model descriptions using generic rules, which have been implemented using model queries. If some rule is not followed, an information message will be shown to the user.
  - **Specification on-demand of the interoperability mechanisms** to only support the concepts of a different modeling language that users need to describe the variation points (user-dependent properties) rather than create interoperability mechanisms for supporting the fullset of the concepts of the two involved modeling languages. Thus, the necessary effort for specifying the interoperability mechanisms can be reduced obtaining an efficient support.

- Relevant results for the Variability Management community.
  - **A novel way to use variability management** in which not only is used to manage variabilities in a base model but also, use these variabilities to enable collaborative modeling from a different modeling language.
  - **The application results**, which reveal that the facet of variability models does not limit the expressiveness of users to a bounded selection of features during the collaborative modeling

as our initial attempt using the facet of features for managing variabilities does.

We hope that these contributions and results encourage researchers and practitioners to apply our approach to other promising areas of research and industry.

10.2 Assessment and Future Work

The work presented so far reveals insights combining the End-user Development, the Model-Driven Development and the Variability Management fields to involve users in the modeling effort. Although our work tackled challenges and ideas of these fields, a close assessment is necessary to reveal some limitations of this work and propose some future work.

We believe that using our model-driven and variability-based approach is a promising way to integrate model descriptions that have been performed using a different modeling language. This approach brings the following important benefits: (1) users are able to participate in the modeling effort using a different modeling language; (2) different modeling languages are able to interoperate in order to obtain the full description of a software system; (3) users can be focused on describing the concerns of the software system that they are experts with rather than describe the entire software system; and (4) the structure of the modeling languages is not modified, which promotes the application of the approach in existing MDD processes.

The main innovation of our approach is the combination of both variability and modeling techniques to (1) delimit the concerns that may be described by a different user by creating gaps and completing them, and (2) reuse the specification of the approach if the descriptions of concerns change. We believe that our approach is especially useful for both taking advantage of

the experience of different users in modeling tasks and preventing those roles are forced to use different modeling primitives in order to participate in the same project.

Although our approach automatically supports collaborative modeling descriptions in a transparent way for users once it is specified, the specification of the approach by the modeler expert requires the most-time consuming (e.g., to specify the correspondences in the weaving model and transformation rules) as the three different applications of our approach have revealed. Hence, some aspects of the tool support have to be improved in our approach to reduce the necessary effort to initialize our approach. For example, we believe that the extension of our approach contributes towards an efficient definition of the interoperability mechanisms since they are defined on-demand to only support the concepts that are involved in collaborative modeling descriptions rather than defining interoperability mechanisms to the fullset of concepts.

The technological decisions that support the main building blocks of our approach (collaborative modeling and interoperability mechanisms) are aligned with current modeling standards and MDD-oriented technologies. In particular, we have proposed both MOF meta-models as stating point in our approach to support non-intrusive interoperability of any DSL based on MOF, and CVL variability models (an standard proposal for the OMG) to support collaborative modeling.

These technological decisions may favor that researchers and practitioners apply our approach to other existing modeling approaches that are based on MOF. Nevertheless, we believe that these decisions present some limitations in the application of our approach in some cases (e.g., users that use a language that is not based on MOF, or some required inputs to apply our approach such as the Ecore meta-models of the involved modeling approaches are not provided).

Therefore, the research presented here is not a closed work and there are several interesting directions that can be taken to provide the proposal with a wider spectrum of application and reduce the required specification effort. The following list offers several interesting activities to continue this work.

Generic model-to-model transformations. Our goal is to provide generic model-to-model transformation rules that work as easy as a language translator by only selecting the weaving model (that bridges the meta-model concepts of the two different modeling languages) and the source model as input to create a target model as output. Thus, the Modeler expert does not have to create the transformation rules and they will be also reusable in different modeling languages. Although this future improvement is out of scope of this thesis work, we started working on considering the best solution but we have found some problems with model heterogeneities.

Model validation. We integrated model validation mechanisms in the toolkit by means of a set of domain-independent rules using EMFMQ, since we detected some problems during collaborative modeling descriptions (i.e., user-dependent descriptions do not fit the boundaries of the variability model). Although we address a great number of problems, we believe that it is important to provide more validation mechanisms for checking syntactic and semantic problems in models (i.e., user-dependent properties may cause conflicts with the base model properties). Hence, further work is needed to add more domain-independent rules for model validation, or to combine the rules with other existing model validation mechanisms such as the Epsilon Validation Language [175].

Tool support. Some technologies were used to implement the toolkit that

supports our ideas (i.e., EMF, EMFMQ, CVL, etc.). Although we spent a great amount of effort on it to work, still the toolkit requires further work to be used by a regular modeler expert (i.e., a user-friendly interface, guidance of the steps that should be followed, guidance of the inputs that should be provided, etc.).

Variability management concrete syntax. We detected in the different applications of our approach that some design decisions have to be taken about the concrete syntax of the variability management language concepts (i.e., how the *replacement fragment* and *boundaries* of CVL are represented), so we believe that more work is needed to set guidelines for the concrete syntax of these concepts.

Multi-user support. The participation of more than two users in a common modeling project may be necessary in some scenarios (i.e., a smart home modeling project in which more than one user should be involved in the specification of the user-dependent properties). Although we addressed multi-user support by managing users and policies in our first attempt for supporting collaborative modeling using features [140], it requires further work in our approach using variability models to deal with conflicts with regard to: users descriptions, management of users, ownership of objects or skills, user descriptions from more than one different modeling approach since our approach can be applied more than once in the same base modeling language to support users with different skills (i.e., in a smart home modeling project, users described their dependent-properties as we shown but also, it is necessary to involve in the same modeling project an electrician who uses a different closer language to specify the devices of the home and how they are connected).

10.3 Publications

The research activity presented in this work has been presented and discussed before on different peer-review forums. The distinct publications are ordered by year of publication and the author position is used as an indicator of the degree of contribution made by the author of this thesis was involved. The publications are the following:

- **Francisca Pérez**, Pedro Valderas and Joan Fons. *Collaborative Modeling through the Integration of Heterogeneous Modeling Languages*. 22nd International Conference on Information Systems Development (ISD2013). Sevilla, Spain, 2013.

- **Francisca Pérez**, Pedro Valderas and Joan Fons. *A Domain-Specific Language for Enabling Doctors to Specify Biomechanical Protocols*. IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2013). San Jose, CA, USA, 2013.

- **Francisca Pérez**, Pedro Valderas and Joan Fons. *Allowing End-users to Participate within Model-Driven Development Approaches*. IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2011). Pittsburgh, PA, USA, 2011.

- **Francisca Pérez**, Pedro Valderas and Joan Fons. *Towards the Involvement of End-users within Model-Driven Development*. Third International Symposium on End-User Development (IS-EUD 2011). Torre Canne (Brindisi), Italy, 2011.

- Estefanía Serral, **Francisca Pérez**, Pedro Valderas, Vicente Pelechano. *An End-User Tool for Adapting Home Automation to User Behaviour*

at Runtime. IV International Conference on Ubiquitous Computing and Ambient Intelligence (UCAmI 2010). Valencia, SPAIN, 2010.

- **Francisca Pérez** and Pedro Valderas. *Allowing End-users to Actively Participate within the Elicitation of Pervasive System Requirements through Immediate Visualization*. Fourth International Workshop on Requirements Engineering Visualization (REV'09). Atlanta, Georgia, USA. 2009.

- **Francisca Pérez**, Carlos Cetina, Pedro Valderas and Joan Fons. *Towards End-User Development of Smart Homes by means of Variability Engineering*. Third International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'09). Sevilla, Spain. 2009.

- **Francisca Pérez**, Pedro Valderas and Joan Fons. *Enabling End-users Participation in an MDD-SPL Approach*. 1st International Workshop on Model-driven Approaches in Software Product Line Engineering (MAPLE 2009). San Francisco, California, USA. 2009.

- **Francisca Pérez** and Pedro Valderas. *A Tool-supported Natural Requirements Elicitation Technique for Pervasive Systems centred on End-users*. XIV Jornadas de Ingeniería del Software y Bases de Datos. San Sebastián, Spain. 2009.

10.3.1 Relevance of the publications

This section provides some information about the relevance of some of the publications presented above where different aspects of this work have been published.

ISD. According to the CORE conference ranking, the International Conference on Information Systems Development (ISD) is Tier-A. It is

recognized as being one of the most important conferences in the area of information systems engineering.

VL/HCC. According to the CORE conference ranking, the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC) is Tier-A. It is recognized as one of the most important conference in the area of visual languages and human-centric computing.

REV. According to the CORE conference ranking, the International Workshop on Requirements Engineering Visualization (REV) is Tier-B. It is recognized as an important workshop related to the area of visualization of requirements and ways of making them practical.

UCAmI. The Ubiquitous Computing and Ambient Intelligence (UCAmI) conference has been consolidated as a reference event in Ubiquitous Computing & Ambient Intelligence, agglutinating high quality papers. This conference provides a discussion forum where researchers and practitioners on Ubiquitous Computing and Ambient Intelligence can meet, disseminate and exchange ideas and problems, identify some of the key issues related to these topics, and explore together possible solutions and future works.

IS-EUD. The International Symposium on End-User Development (IS-EUD) is an important international symposium to bring together researchers and practitioners from industry and academia that are working in the field of End-user Development.

International workshops. In addition to the above mentioned venues, different parts of the work have been published in workshops from relevant conferences such as the International Software Product Line Conference (SPLC). This has helped to achieve diffusion of the work.

10.4 Projects Directed

In addition, one degree project was directed and one degree project was co-directed in the context of this work to explore some concepts and put into practice its application. They are listed as follows:

- *Herramienta para transformar modelos mediante reglas genéricas y reutilizables*. Alejandro Del Ruste Palau. Universidad San Jorge. July 2013.
- *Implementación de un editor de usuarios finales para la configuración de su entorno en un hogar digital*. Hugo Ricós Llorca. Universitat Politècnica de València. October 2011.

10.5 Final Conclusion

Henry Ford, founder of the car company that bears his name, revolutionized the automotive industry and converted the automobile from an expensive curiosity into a practical conveyance that would profoundly impact the landscape of the twentieth century. It was Henry Ford who stated:

"Coming together is a beginning, staying together is progress, and working together is success."

– Henry Ford (1863-1947)

This quote emphasizes the importance of collaborate with others to successfully reach one goal. Specifically, it can be transferred to Model-Driven-Development processes in which the involved parties may work together to the success of the modeling project.

Nevertheless, some involved parties such as users are usually interviewed or in other ways heard but they lack the skills to actively participate in

modeling tasks. Therefore, it becomes increasingly essential giving users the opportunity to contribute themselves in model descriptions. The involvement of users from the very beginning could help to achieve a wider adoption of Model-Driven-Development processes. In the particular case of involving end-users, it creates a sense of ownership of the process that makes results more difficult to reject in the future, and minimize the mismatch between end-user expectations and system behavior. This thesis is an attempt to achieve *"working together is success"* in Model-Driven Development by actively involving users in modeling tasks.

BIBLIOGRAPHY

[1] Gunter Mussbacher, Daniel Amyot, Ruth Breu, Jean-Michel Bruel, Betty Cheng, Philippe Collet, Benoit Combemale, Robert France, Rogardt Heldal, James Hill, Jörg Kienzle, Matthias Schöttle, Friedrich Steimann, Dave Stikkolorum, and Jon Whittle. The Relevance of Model-Driven Engineering Thirty Years from Now. In *Model-Driven Engineering Languages and Systems*, volume 8767 of *Model-Driven Engineering Languages and Systems*, page 18, Valencia, Spain, September 2014. Springer International Publishing Switzerland.

[2] J. Whittle, J. Hutchinson, and M. Rouncefield. The state of practice in model-driven engineering. *Software, IEEE*, 31(3):79–85, May 2014.

[3] Ivan Mistrík, John Grundy, André Hoek, and Jim Whitehead. *Collaborative Software Engineering: Challenges and Prospects*. Springer Berlin Heidelberg, 2010.

[4] Benoit Combemale, Julien Deantoni, Benoit Baudry, Robert France, Jean-Marc Jézéquel, and Jeff Gray. Globalizing Modeling Languages. *Computer*, pages 68–71, June 2014.

[5] Alexey Voinov and Francois Bousquet. Modelling with stakeholders. *Environmental Modelling Software*, 25(11):1268 – 1281, 2010.

[6] Michiel Renger, Gwendolyn L. Kolfschoten, and Gert-Jan de Vreede. Challenges in collaborative modeling: A literature review. In *Advances*

in Enterprise Engineering I, held at CAiSE 2008, volume 10, pages 61–77, Montpellier, France, 2008.

[7] Christoph Neumann, Ronald A. Metoyer, and Margaret Burnett. End-user strategy programming. *J. Vis. Lang. Comput.*, 20(1):16–29, 2009.

[8] *EUSES Consortium.* http://eusesconsortium.org, 2013.

[9] Stephen J. Mellor, Anthony N. Clark, and Takao Futagami. Guest editors' introduction: Model-driven development. In *IEEE Software*, page 20(5):14:18, 2003.

[10] Jean-Marc Jézéquel. Model driven design and aspect weaving. *Journal of Software and Systems Modeling (SoSyM)*, 7(2):209–218, may 2008.

[11] Krzysztof Gajos, Harold Fox, and Howard Shrobe. End user empowerment in human centered pervasive computing. In *Proceedings of Pervasive 2002*, pages 1–7, 2002.

[12] Jan Humble, Andy Crabtree, Terry Hemmings, Karl-Petter kesson, Boriana Koleva, Tom Rodden, and Pär Hansson. "playing with the bits" user-configuration of ubiquitous domestic environments. In *UbiComp 2003: Ubiquitous Computing*, volume 2864 of *Lecture Notes in Computer Science*, pages 256–263. Springer Berlin Heidelberg, 2003.

[13] Maria Francesca Costabile, Piero Mussio, Loredana Parasiliti Provenza, and Antonio Piccinno. End users as unwitting software developers. In *WEUSE '08: Proceedings of the 4th international workshop on End-user software engineering*, pages 6–10, New York, NY, USA, 2008. ACM.

[14] Henry Lieberman, Fabio Paterno, and Volker Wulf. *End User Development.* Springer, 2006.

[15] Muneera Bano and Didar Zowghi. A systematic review on the relationship between user involvement and system success. *Information and Software Technology*, 58(0):148 – 169, 2015.

[16] Deepak Dhungana, Dominik Seichter, Goetz Botterweck, Rick Rabiser, Paul Grünbacher, David Benavides, and José Galindo. Configuration of multi product lines by bridging heterogeneous variability modeling approaches. In *Software Product Line Conference (SPLC), 2011 15th International*, pages 120–129, Aug 2011.

[17] Mary Beth Rosson, Hansa Sinha, Mithu Bhattacharya, and Dejin Zhao. Design planning by end-user web developers. *J. Vis. Lang. Comput.*, 19(4):468–484, 2008.

[18] Hugo Brunelière, Jordi Cabot, Cauê Clasen, Frédéric Jouault, and Jean Bézivin. Towards Model Driven Tool Interoperability: Bridging Eclipse and Microsoft Modeling Tools. In *Modelling Foundations and Applications*, volume 6138, chapter 5, pages 32–47. 2010.

[19] Giovanni Giachetti, Beatriz Marín, and Oscar Pastor. Using uml as a domain-specific modeling language: A proposal for automatic generation of uml profiles. In *Proceedings of the 21st International Conference on Advanced Information Systems Engineering*, CAiSE '09, pages 110–124, Berlin, Heidelberg, 2009. Springer-Verlag.

[20] Andreas L. Opdahl. Incorporating uml class and activity constructs into ueml. In *ER Workshops*, volume 6413, pages 244–254. Springer, 2010.

[21] Felix Klar, Sebastian Rose, and Andy Schürr. A meta-model-driven tool integration development process. volume 5 of *Lecture Notes in Business Information Processing*, pages 201–212. Springer, 2008.

[22] Juan Manuel Vara and Esperanza Marcos. A framework for model-driven development of information systems: Technical decisions and lessons learned. *Journal of Systems and Software*, 85(10):2368 – 2384, 2012.

[23] Xavier Blanc, Marie-Pierre Gervais, and Prawee Sriplakich. Model bus: Towards the interoperability of modelling tools. In *Model Driven Architecture*, volume 3599 of *Lecture Notes in Computer Science*, pages 17–32. Springer Berlin Heidelberg, 2005.

[24] Marcos Didonet Del Fabro, Jean Bézivin, and Patrick Valduriez. Model-driven tool interoperability: An application in bug tracking. In *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*, volume 4275 of *Lecture Notes in Computer Science*, pages 863–881. Springer Berlin Heidelberg, 2006.

[25] Salvatore T. March and Gerald F. Smith. Design and natural science research on information technology. *Decis. Support Syst.*, 15(4):251–266, 1995.

[26] Vijay Vaishnavi and William Kuechler. Design research in information systems. http://www.isworld.org/Researchdesign/drisISworld.htm, January 2004.

[27] Henry Lieberman, Fabio Paternò, Markus Klann, and Volker Wulf. End-user development: An emerging paradigm. In *End User Development*, volume 9 of *Human-Computer Interaction Series*, chapter 1, pages 1–8. Springer Netherlands, Dordrecht, 2006.

[28] Alexander Repenning and Andri Ioannidou. What makes end-user development tick? 13 design guidelines. In *End User Development*,

volume 9 of *Human-Computer Interaction Series*, pages 51–85. Springer Netherlands, 2006.

[29] Maria F. Costabile, Daniela Fogli, Catherine Letondal, Piero Mussio, and Antonio Piccinno. Domain-expert users and their needs of software development. In *Interaction*, volume 4, pages 532–536, 2003.

[30] Anind K. Dey, Raffay Hamid, Chris Beckmann, Ian Li, and Daniel Hsu. A cappella: programming by demonstration of context-aware applications. In *CHI '04*, pages 33–40, New York, USA, 2004.

[31] Brad A. Myers, John F. Pane, and Andy Ko. Natural programming languages and environments. *Commun. ACM*, 47(9):47–52, September 2004.

[32] Henry Lieberman. Programming by example (introduction). *Commun. ACM*, 43(3):72–74, 2000.

[33] Graham Clarke Jeannette Shiaw-Yuan Chin, Victor Callaghan. A programming-by-example approach to customising digital homes. *Intelligent Environments, 2008 IET 4th International Conference on*, pages 1–8, July 2008.

[34] Graham Clarke Jeannette Shiaw-Yuan Chin, Victor Callaghan. An end-user programming paradigm for pervasive computing applications. *International Conference on Pervasive Services*, pages 325–328, 2006.

[35] Alexander Repenning and Corrina Perrone. Programming by example: programming by analogous examples. *Commun. ACM*, 43(3):90–97, 2000.

[36] Andrew J. Ko and Brad A. Myers. Designing the whyline: a debugging interface for asking questions about program behavior. In *CHI '04 proceedings*, pages 151–158, New York, USA, 2004. ACM Press.

[37] Brad A. Myers. Taxonomies of visual programming and program visualization. *J. Vis. Lang. Comput.*, 1(1):97–123, March 1990.

[38] Khai N. Truong, Elaine M. Huang, and Gregory D. Abowd. Camp: A magnetic poetry interface for end-user programming of capture applications for the home. In *in Proceedings of Ubicomp 2004*, pages 143–160, 2004.

[39] Vernica Andrea Bollati, Juan Manuel Vara, Álvaro Jiménez, and Esperanza Marcos. Applying mde to the (semi-)automatic development of model transformations. *Information and Software Technology*, 55(4):699 – 718, 2013.

[40] OMG. MDA Guide Version 1.0.1. `http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf`, June 2003.

[41] Stuart Kent. Model driven engineering. In *Proceedings of the Third International Conference Integrated Formal Methods (IFM'2002)*, 2002.

[42] Aditya Agrawal, Tihamer Levendovszky, Jon Sprinkle, Feng Shi, and Gabor Karsai. Generative programming via graph transformations in the model-driven architecture. In *In OOPSLA 2002 Workshop in Generative Techniques in the context of Model Driven Architecture*, 2002.

[43] Bran Selic. The pragmatics of model-driven development. *IEEE Softw.*, 20(5):19–25, 2003.

[44] Amílcar Sernadas, Cristina Sernadas, and Hans-Dieter Ehrich. Object-oriented specification of databases: An algebraic approach. In *VLDB '87: Proceedings of the 13th International Conference on Very Large Data Bases*, pages 107–116, San Francisco, CA, USA, 1987. Morgan Kaufmann Publishers Inc.

[45] Ralf Jungclaus, Gunter Saake, Thorsten Hartmann, and Cristina Sernadas. TROLL: a language for object-oriented specification of information systems. *ACM Trans. Inf. Syst.*, 14(2):175–211, 1996.

[46] Michael Rohs and Jürgen Bohn. Entry points into a smart campus environment.overview of the ethoc system. In *ICDCSW '03: Proceedings of the 23rd International Conference on Distributed Computing Systems*, page 260, Washington, DC, USA, 2003. IEEE Computer Society.

[47] Object Management Group. Unified modeling language: Superstructure version 2.1.1. OMG Specification, February 2007.

[48] Philippe Kruchten. *The Rational Unified Process: An Introduction.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[49] Ivan Kurtev Ivanov. *Adaptability of Model Transformations.* phdthesis, IPA, 2005. ISBN 90-365-2184-X.

[50] Jean Bézivin, Nicolas Farcet, Jean-Marc Jézéquel, Benoît Langlois, and Damien Pollet. Reflective model driven engineering. pages 175–189. Springer, 2003.

[51] Jean Bézivin. In search of a basic principle for model driven engineering. *Novatica Journal, Special Issue*, 5(2):21–24, 2004.

[52] Douglas C. Schmidt. Guest editor's introduction: Model-driven engineering. *Computer*, 39(2):25–31, 2006.

[53] Robert Balzer. A 15 year perspective on automatic programming. *IEEE Trans. Softw. Eng.*, 11(11):1257–1268, 1985.

[54] Krzysztof Czarnecki. *Generative Programming. Principles and Techniques of Software Engineering Based on Automated Conguration and Fragment-Based Component Models.* PhD thesis, Technical University of Ilmenau, October 1998.

[55] Ulrich W. Eisenecker. Generative programming (gp) with c++. In *JMLC '97: Proceedings of the Joint Modular Languages Conference on Modular Programming Languages*, pages 351–365, London, UK, 1997. Springer-Verlag.

[56] Arie van Deursen, Paul Klint, and Joost Visser. Domain-specific languages: an annotated bibliography. volume 35, pages 26–36, New York, NY, USA, 2000. ACM.

[57] Martin Fowler. *Domain Specific Languages.* Addison-Wesley Professional, 1st edition, 2010.

[58] Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and how to develop domain-specific languages. *ACM Comput. Surv.*, 37(4):316–344, December 2005.

[59] Juha pekka Tolvanen and Steven Kelly. Modelling languages for product families a method engineering approach. In *1st OOPSLA Workshop on Domain-specic Visual Languages*, 2001.

[60] Io Ns and M. Simos. *Organization Domain Modeling and OO Analysis and Design: Distinctions, Integration, New Directions.* 1997.

[61] Robert Esser and Jörn W. Janneck. A framework for defining domain-specific visual languages. In *OOPSLA 2001 Workshop on Domain Specific Visual Languages*, 2001.

[62] Thomas Stahl, Markus Völter, Jorn Bettin, Arno Haase, and Simon Helsen. *Model-driven software development - technology, engineering, management*. Pitman, 2006.

[63] Object Management Group. Meta object facility (MOF) specification, 2000. `http://www.omg.org`.

[64] Markus Völter. Md* best practices. *Journal of Object Technology*, 8(6):79–102, 2009.

[65] Esther Guerra, Juan de Lara, Alessio Malizia, and Paloma Daz. Supporting user-oriented analysis for multi-view domain-specific visual languages. *Information and Software Technology*, 51(4):769 – 784, 2009.

[66] Anne Geraci. *IEEE Standard Computer Dictionary: Compilation of IEEE Standard Computer Glossaries*. IEEE Press, Piscataway, NJ, USA, 1991.

[67] *ATL Transformation Language*. `https://www.eclipse.org/atl/`, 2015.

[68] *Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT)*. `http://www.omg.org/spec/QVT/`, 2015.

[69] Hartmut Ehrig, Karsten Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation (Monographs in Theoretical Computer Science. An EATCS Series)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[70] Andy Schürr. Specification of graph translators with triple graph grammars. In *Graph-Theoretic Concepts in Computer Science*, volume 903 of *Lecture Notes in Computer Science*, pages 151–163. Springer Berlin Heidelberg, 1995.

[71] Frank Legler and Felix Naumann. A classification of schema mappings and analysis of mapping tools. In *BTW*, volume 103 of *LNI*, pages 449–464. GI, 2007.

[72] Anas Abouzahra, Jean Bézivin, Marcos Didonet Del Fabro, and Frédéric Jouault. A practical approach to bridging domain specific languages with uml profiles. In *Proceedings of the Best Practices for Model Driven Software Development at OOPSLA*, volume 5. Citeseer, 2005.

[73] Albin Jossic, Marcos Didonet Del Fabro, J-P Lerat, Jean Bézivin, and Frédéric Jouault. Model integration with model weaving: a case study in system architecture. In *Systems Engineering and Modeling, 2007. ICSEM '07. International Conference on*, pages 79–84, March 2007.

[74] Marcos López-Sanz, Juan Manuel Vara, Esperanza Marcos, and Carlos E. Cuesta. A model-driven approach to weave architectural styles into service-oriented architectures. In *Proceedings of the First International Workshop on Model Driven Service Engineering and Data Quality and Security*, MoSE+DQS '09, pages 53–60, New York, NY, USA, 2009. ACM.

[75] Jesús Sánchez Cuadrado, Esther Guerra, and Juan Lara. Generic model transformations: Write once, reuse everywhere. In *Theory and Practice of Model Transformations*, volume 6707 of *Lecture Notes in Computer Science*, pages 62–77. 2011.

[76] Marcos Didonet Del Fabro and Patrick Valduriez. Towards the efficient development of model transformations using model weaving and matching transformations. *Software & Systems Modeling*, 8(3):305–324, 2009.

[77] Muhammad Ali Babar, Lianping Chen, and Forrest Shull. Managing variability in software product lines. *IEEE Software*, 27(3):89–91, 2010.

[78] Marco Sinnema and Sybren Deelstra. Classifying variability modeling techniques. *Inf. Softw. Technol.*, 49(7):717–739, July 2007.

[79] Sven Apel, Florian Janda, Salvador Trujillo, and Christian Kstner. Model superimposition in software product lines. In RichardF. Paige, editor, *Theory and Practice of Model Transformations*, volume 5563 of *Lecture Notes in Computer Science*, pages 4–19. Springer Berlin Heidelberg, 2009.

[80] Jan Bosch. *Design and Use of Software Architectures: Adopting and Evolving a Product-line Approach*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.

[81] Paul C. Clements and Linda Northrop. *Software Product Lines: Practices and Patterns*. SEI Series in Software Engineering. Addison-Wesley, August 2001.

[82] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[83] Frank J. van der Linden, Klaus Schmid, and Eelco Rommes. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[84] Jan Bosch, Gert Florijn, Danny Greefhorst, Juha Kuusela, J. Obbink, and Klaus Pohl. Variability Issues in Software Product Lines Software Product-Family Engineering. In Frank van der Linden, editor, *Software Product-Family Engineering*, volume 2290 of *Lecture Notes in Computer Science*, chapter 3, pages 303–338. Springer Berlin / Heidelberg, Berlin, Heidelberg, April 2002.

[85] Lianping Chen, Muhammad Ali Babar, and Nour Ali. Variability management in software product lines: A systematic review. In *Proceedings of the 13th International Software Product Line Conference*, SPLC '09, pages 81–90, Pittsburgh, PA, USA, 2009. Carnegie Mellon University.

[86] Marco Aiello, Pavel Bulanov, and Heerko Groefsema. Requirements and tools for variability management. In *COMPSAC Workshops*, pages 245–250. IEEE Computer Society, 2010.

[87] Klaus Schmid and Isabel John. A customizable approach to full lifecycle variability management. *Sci. Comput. Program.*, 53(3):259–284, December 2004.

[88] Felix Bachmann and Len Bass. Managing variability in software architectures. In *Proceedings of the 2001 Symposium on Software Reusability: Putting Software Reuse in Context*, SSR '01, pages 126–132, New York, NY, USA, 2001. ACM.

[89] Markus Voelter and Iris Groher. Product line implementation using aspect-oriented and model-driven software development. *Software Product Line Conference, 2007. SPLC 2007. 11th International*, pages 233–242, 10-14 Sept. 2007.

[90] Øystein Haugen, Birger Møller-Pedersen, Jon Oldevik, Gøran K. Olsen, and Andreas Svendsen. Adding standardized variability to domain specific languages. In *SPLC '08*, pages 139–148, Washington, DC, USA, 2008. IEEE Computer Society.

[91] Kyo C. Kang, Sajoong Kim, Jaejoon Lee, Kijoo Kim, Euiseob Shin, and Moonhang Huh. *Feature-Oriented Domain Analysis (FODA) Feasibility Study.* 1990.

[92] Kyo Chul Kang and Hyesun Lee. Variability modeling. In *Systems and Software Variability Management*, pages 25–42. 2013.

[93] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. Automated analysis of feature models 20 years later: A literature review. *Inf. Syst.*, 35(6):615–636, September 2010.

[94] Andreas Svendsen, Øystein Haugen, and Xiaorui Zhang. *Cvl 1.2 user guide*, 2011.

[95] Markus Voelter and Eelco Visser. Product line engineering using domain-specific languages. In *Software Product Line Conference (SPLC), 2011 15th International*, pages 70–79, 2011.

[96] Object Management Group. *Variability Modeling.* `http://www.omgwiki.org/variability/doku.php?id=introduction_to_variability_modeling`, 2009.

[97] *Common Variability Language Revised Submission.* `http://www.omgwiki.org/variability/doku.php`, 2012.

[98] Franck Fleurey, Øystein Haugen, Birger Møller-Pedersen, Gøran K. Olsen, Andreas Svendsen, and Xiaorui Zhang. A Generic Language and Tool for Variability Modeling. Technical report, 2009.

[99] Javier Muñoz. *Model Driven Development of Pervasive Systems. Building a Software Factory.* Phd thesis, Universidad Politécnica de Valencia, 2008.

[100] Manuel Jimenez, Francisca Rosique, Pedro Sanchez, Barbara Alvarez, and Andres Iborra. Habitation: A domain-specific language for home automation. *IEEE Software*, pages 30–38, 2009.

[101] Jane Webster and Richard T. Watson. Analyzing the past to prepare for the future: Writing a literature review. *MIS Q.*, 26(2):xiii–xxiii, June 2002.

[102] Khai N. Truong and Gregory D. Abowd. Inca: A software infrastructure to facilitate the construction and evolution of ubiquitous capture access applications. In *Pervasive*, volume 3001 of *Lecture Notes in Computer Science*, pages 140–157. Springer, 2004.

[103] Krzysztof Gajos. Rascal - a resource manager for multi agent systems in smart spaces. In *Revised Papers from the Second International Workshop of Central and Eastern Europe on Multi-Agent Systems: From Theory to Practice in Multi-Agent Systems*, CEEMAS '01, pages 111–120, London, UK, UK, 2002. Springer-Verlag.

[104] Ajay Kulkarni. *A Reactive Behavioral System for the Intelligent Room.* Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 2002.

[105] Robin Abraham, Margaret Burnett, and Martin Erwig. Spreadsheet programming. *Wiley Encyclopedia of Computer Science and Engineering*, 2008.

[106] Margaret Burnett, Gregg Rothermel, and Curtis Cook. Software engineering for end-user programmers. *In Proceedings of the CHI'03 Workshop on End-User Development*, 2003.

[107] Margaret Burnett, Curtis Cook, and Gregg Rothermel. End-user software engineering. *Commun. ACM*, 47(9):53–58, 2004.

[108] Gregg Rothermel, Margaret Burnett, Lixin Li, Christopher Dupuis, and Andrei Sheretov. A methodology for testing spreadsheets. *ACM Trans. Softw. Eng. Methodol.*, 10(1):110–147, January 2001.

[109] Marc Fisher II, Gregg Rothermel, Darren Brown, Mingming Cao, Curtis R. Cook, and Margaret M. Burnett. Integrating automated test generation into the wysiwyt spreadsheet testing methodology. *ACM TRANS. SOFTW. ENG. METHODOL*, 15:2006, 2006.

[110] Chong Wang, Miao Xiong, Qi Zhou, and Yong Yu. Panto: A portable natural language interface to ontologies. In *The Semantic Web: Research and Applications*, volume 4519 of *Lecture Notes in Computer Science*, pages 473–487. Springer Berlin Heidelberg, 2007.

[111] Markus Voelter. Language and ide modularization and composition with mps. In *Generative and Transformational Techniques in Software Engineering IV*, volume 7680 of *Lecture Notes in Computer Science*, pages 383–430. Springer Berlin Heidelberg, 2013.

[112] *JetBrains' Meta Programming System*. `http://jetbrains.com/mps`, 2014.

[113] Giovanni Andrés Giachetti Herrera. *Supporting Automatic Interoperability in Model-Driven Development Processes*. PhD thesis, Universitat Politècnica de València, 2011.

[114] Esther Guerra, Juan de Lara, and Fernando Orejas. Inter-modelling with patterns. *Software and System Modeling*, 12(1):145–174, 2013.

[115] Gerti Kappel, Manuel Wimmer, Werner Retschitzegger, and Wieland Schwinger. Leveraging model-based tool integration by conceptual modeling techniques. In *The Evolution of Conceptual Modeling*, volume 6520 of *Lecture Notes in Computer Science*, pages 254–284. Springer Berlin Heidelberg, 2011.

[116] Javier Luis Cánovas Izquierdo and Jordi Cabot. Enabling the collaborative definition of dsmls. In *Advanced Information Systems Engineering*, volume 7908 of *Lecture Notes in Computer Science*, pages 272–287. Springer Berlin Heidelberg, 2013.

[117] The gemoc initiative. `http://gemoc.org`, 2015.

[118] Tobias Hildenbrand, Franz Rothlauf, Michael Geisser, Armin Heinzl, and Thomas Kude. Approaches to collaborative software development. In *CISIS*, pages 523–528, 2008.

[119] Oscar Dieste Dante Carrizo and Natalia Juristo. Study of elicitation techniques adequacy. In *11th Workshop on Requeriments Engineering (WER)*, 2008.

[120] Hugh R. Beyer and Karen Holtzblatt. Apprenticing with the customer. *Commun. ACM*, 38(5):45–52, 1995.

[121] Michael G. Christel;Kyo C. Kang. Issues in requirements elicitation. Technical report, 09/1992 1992. Prepared for the SEI Joint Program Office, HQ ESC/AXS 5 Eglin Street, Hanscom AFB, MA 01731-2116. Sponsored by the U.S. Department of Defense.

[122] Peter Rittgen. Coma: A tool for collaborative modeling. In *CAiSE Forum*, volume 344 of *CEUR Workshop Proceedings*, pages 61–64. CEUR-WS.org, 2008.

[123] Jac A.M. Vennix, David F. Andersen, George P. Richardson, and John Rohrbaugh. Model-building for group decision support: Issues and alternatives in knowledge elicitation. *European Journal of Operational Research*, 59(1):28 – 41, 1992. Modelling for Learning.

[124] Alberto Avritzer and Daniel J. Paulish. A comparison of commonly used processes for multi-site software development. In *Collaborative Software Engineering*, pages 285–302. Springer Berlin Heidelberg, 2010.

[125] Palmer R.N. Lund, Jay R. Water resource system modeling for conflict resolution. In *Water Resources Update 3 (108)*, pages 70–82, 1997.

[126] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Softw. Engg.*, 14(2):131–164, 2009.

[127] Yunwen Ye and Gerhard Fischer. Designing for participation in socio-technical software systems. In *UAHCI'07 proceedings*, pages 312–321, Berlin, Heidelberg, 2007. Springer-Verlag.

[128] Mary Beth Rosson, Hansa Sinha, Mithu Bhattacharya, and Dejin Zhao. Design planning by end-user web developers. *J. Vis. Lang. Comput.*, 19(4):468–484, 2008.

[129] John Steinmetz. Computers and squeak as environments for learning. *Squeak : Open Personal Computing for Multimedia*, 2000.

[130] David Canfield Smith, Allen Cypher, and James C. Spohrer. Kidsim: programming agents without a programming language. *Commun. ACM*, 37(7):54–67, 1994.

[131] Judith Segal. Two principles of end-user software engineering research. In *WEUSE I proceedings*, pages 1–5, New York, NY, USA, 2005. ACM.

[132] Jakob Nielsen. *Usability Engineering*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[133] Martijn van Welie and Hallvard Trætteberg. Interaction patterns in user interfaces. In *Proc. Seventh Pattern Languages of Programs Conference: PLoP 2000*, pages 13–16, 2000.

[134] Mick P. Couper, Roger Tourangeau, Frederick G. Conrad, and Scott D. Crawford. What they see is what we get: response options for web surveys. *Soc. Sci. Comput. Rev.*, 22(1):111–127, 2004.

[135] Wilbert O. Galitz. *The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques*. John Wiley & Sons, Inc., New York, NY, USA, 2002.

[136] Markus Voelter and Iris Groher. Product line implementation using aspect-oriented and model-driven software development. *SPLC 2007*, pages 233–242, Sept. 2007.

[137] Javier Muñoz and Vicente Pelechano. Building a software factory for pervasive systems development. In *CAiSE*, pages 342–356, 2005.

[138] Michalis Anastasopoulos, T. Patzke, and M. Becker. Software product line technology for ambient intelligence applications. In *In Proc. Net.ObjectDays*, pages 179–195, 2005.

[139] Jon O'Brien, Tom Rodden, Mark Rouncefield, and John Hughes. At home with the technology: an ethnographic study of a set-top-box trial. *ACM Trans. Comput.-Hum. Interact.*, 6(3):282–308, 1999.

[140] Francisca Pérez and Pedro Valderas. Allowing end-users to actively participate within the elicitation of pervasive system requirements through immediate visualization. In *REV'09*, pages 31–40, Washington, USA, 2009. IEEE Computer Society.

[141] Francisca Pérez, Pedro Valderas, and Joan Fons. Enabling end-users participation in an mdd-spl approach. In *MAPLE 2009 Workshop*, 2009.

[142] Javier Muñoz and Vicente Pelechano. Applying software factories to pervasive systems: A platform specific framework. In *ICEIS (3)*, pages 337–342, 2006.

[143] Javier Muñoz, Vicente Pelechano, and Carlos Cetina. Implementing a pervasive meeting room: A model driven approach. In *IWUC*, pages 13–20, 2006.

[144] Pierre-Yves Schobbens, Patrick Heymans, Jean-Christophe Trigaux, and Yves Bontemps. Generic semantics of feature diagrams. *Comput. Networks*, 51(2):456–479, 2007.

[145] Antonio Ruiz-Cortés David Benavides, Pablo Trinidad. Automated reasoning on feature models. *LNCS, Advanced Information Systems Engineering: 17th International Conference, CAiSE 2005*, 3520:491–503, 2005.

[146] Antonio Ruiz-cortés Sergio Segura Alberto Jimenez Pablo Trinidad, David Benavides. Fama framework. In *12th Software Product Lines Conference (SPLC)*, 2008.

[147] Marcos Didonet Del Fabro, Jean Bézivin, and Patrick Valduriez. Weaving models with the eclipse amw plugin. In *Eclipse Modeling Symposium, Eclipse Summit Europe 2006*.

[148] OO-Method group. Universitat Politècnica de València. *PervML Home*. `http://www.pros.upv.es/index.php/es/flagship-projects/home`, 2012.

[149] Maria Francesca Costabile, Daniela Fogli, Piero Mussio, and Antonio Piccinno. Visual interactive systems for end-user development: A model-based design methodology. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 37(6):1029–1046, 2007.

[150] Joachim Bayer, Sebastien Gerard, Øystein Haugen, Jason Xabier Mansell, Birger Møller-Pedersen, Jon Oldevik, Patrick Tessier, Jean-Philippe Thibault, and Tanya Widen. Consolidated product line variability modeling. in software product lines. In Springer, editor, *Research Issues in Engineering and Management*, 2006.

[151] Brian Elvesæter, Axel Hahn, Arne-Jørgen Berre, and Tor Neple. Towards an interoperability framework for model-driven development of software systems. In *Interoperability of Enterprise Software and Applications*, pages 409–420. Springer London, 2006.

[152] Francisca Pérez, Pedro Valderas, and Joan Fons. Allowing end-users to participate within model-driven development approaches. In *IEEE Symposium on Visual Languages and Human-Centric Computing. VL/HCC 2011*, pages 187–190, 2011.

[153] David Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework 2.0*. Addison-Wesley Professional, 2nd edition, 2009.

[154] Object Management Group (OMG). Xml metadata interchange (xmi) specification, 2014.

[155] Øystein Haugen, Birger Møller-Pedersen, Gøran K. Olsen, Andreas Svendsen, Franck Fleurey, and Xiaorui Zhang. Consolidated CVL language and tool. Technical report, 2010.

[156] Marcos Didonet, Del Fabro, Jean Bézivin, and Patrick Valduriez. Weaving models with the eclipse amw plugin. In *Eclipse Modeling Symposium, Eclipse Summit Europe*, 2006.

[157] Frédéric Jouault, Freddy Allilaire, Jean Bézivin, and Ivan Kurtev. Atl: A model transformation tool. *Sci. Comput. Program.*, 72(1-2):31–39, June 2008.

[158] *Eclipse EMF Model Query Website.* `http://www.eclipse.org/modeling/emf/downloads/?project=query`, 2014.

[159] Zoé Drey and Charles Consel. A Visual, Open-Ended Approach to Prototyping Ubiquitous Computing Applications. In *Proceedings of the 8th IEEE Conference on Pervasive Computing and Communications (PERCOM'10)*, Mannheim Allemagne, 03 2010.

[160] Valencian Regional Ministry of Infrastructure. *UIM basic concepts.* `http://www.moskitt.org/fileadmin/conselleria/documentacion/Manual_Usuario/1.1.3/ManualMetodologicoUIMIngles.pdf`, 2011.

[161] Valencian Regional Ministry of Infrastructure. *Sketcher plugin.* `http://www.moskitt.org/eng/moskitt-1320/`, 2011.

[162] Francisca Pérez, Pedro Valderas, and Joan Fons. Collaborative modeling through the integration of heterogeneous modeling languages.

In *Information System Development*, pages 385–396. Springer International Publishing, 2014.

[163] Francisca Pérez, Pedro Valderas, and Joan Fons. A domain-specific language for enabling doctors to specify biomechanical protocols. In *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 99–102, 2013.

[164] *BTS Bioengineering Kinematics*. `http://www.btsbioengineering.com/BTSBioengineering/Kinematics/BTSSMARTANALYZER/BTS_SMART_ANALYZER.html`, 2013.

[165] Andrew J. Ko, Brad A. Myers, and Htet Htet Aung. Six learning barriers in end-user programming systems. In *Proceedings of the 2004 IEEE Symposium on Visual Languages - Human Centric Computing*, VLHCC '04, pages 199–206, Washington, DC, USA, 2004. IEEE Computer Society.

[166] Martijn van Welie and Hallvard Trætteberg. Interaction patterns in user interfaces. In *PLoP 2000*, pages 13–16, 2000.

[167] Haohai Ma, Weizhong Shao, Lu Zhang 0023, and Yanbing Jiang. Applying OO metrics to assess UML meta-models. In *Proceedings of MODELS/UML'2004*. UML 2004, 2004.

[168] Philip Langer, Manuel Wimmer, and Gerti Kappel. Model-to-model transformations by demonstration. In *Theory and Practice of Model Transformations*, volume 6142 of *Lecture Notes in Computer Science*, pages 153–167. Springer Berlin Heidelberg, 2010.

[169] Dániel Varró. Model transformation by example. In *Model Driven Engineering Languages and Systems*, volume 4199 of *Lecture Notes in Computer Science*, pages 410–424. Springer Berlin Heidelberg, 2006.

[170] Manuel Wimmer, Michael Strommer, Horst Kargl, and Gerhard Kramler. Towards model transformation generation by-example. In *Proceedings of the 40th Annual Hawaii International Conference on System Sciences*, HICSS '07, pages 285b–, Washington, DC, USA, 2007. IEEE Computer Society.

[171] Gerti Kappel, Philip Langer, Werner Retschitzegger, Wieland Schwinger, and Manuel Wimmer. Conceptual modelling and its theoretical foundations. chapter Model Transformation By-example: A Survey of the First Wave, pages 197–215. Springer-Verlag, Berlin, Heidelberg, 2012.

[172] Yu Sun. Model transformation by demonstration. In *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications*, OOPSLA '09, pages 831–832, New York, NY, USA, 2009. ACM.

[173] Iván García-Magariño, Jorge J. Gómez-Sanz, and Rubín Fuentes-Fernández. Model transformation by-example: An algorithm for generating many-to-many transformation rules in several model transformation languages. In RichardF. Paige, editor, *Theory and Practice of Model Transformations*, volume 5563 of *Lecture Notes in Computer Science*, pages 52–66. Springer Berlin Heidelberg, 2009.

[174] Robson do Nascimento Fidalgo, Edson Alves, Sergio España, Jaelson Castro, and Oscar Pastor. Metamodeling the enhanced entity-

relationship model. *Journal of Information and Data Management*, 4(3):406–420, 2013.

[175] Dimitrios S. Kolovos, Richard F. Paige, and Fiona Polack. Detecting and repairing inconsistencies across heterogeneous models. In *Software Testing, Verification, and Validation, 2008 1st International Conference on*, pages 356–364, 2008.

ABOUT

Francisca Pérez is teaching computer science subjects at the *Universidad San Jorge* in which she has a full-time lecturer position from 2012 to nowadays. Although she currently resides in Zaragoza, she was born in Valencia, she studied at the *Universitat Politècnica de València* and she is a researcher at the *Centro de Investigación en Métodos de Producción Software*.

Her research interests are related to Model-Driven Development, Variability Management, End-user Development, and Collaborative Modeling. Her research shows that variability management can be used in a novel way to specify not only variabilities in a base model but also, use these variabilities to enable collaborative modeling from a different modeling language. Thus, different roles such as users can be involved in modeling tasks using a closer modeling language for them.

When she is not glued to a computer screen, she enjoys her family, listening music and traveling. Also, she spends time cooking delicious desserts, dancing and singing in videogames, and trying very hard not be the worst board games player. You can reach her at francisca2perez (at) gmail (dot) com.

UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA