UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

A DOMAIN SPECIFIC LANGUAGE TO SUPPORT THE DEFINITION OF TRANSFORMATION RULES FOR SOFTWARE PROCESS TAILORING

TESIS PARA OPTAR AL GRADO DE
DOCTOR EN CIENCIAS, MENCIÓN COMPUTACIÓN

LUIS GREGORIO SILVESTRE QUIROGA

PROFESORES GUÍA:
MARÍA CECILIA BASTARRICA PIÑEYRO
SERGIO FABIÁN OCHOA DELORENZI

MIEMBROS DE LA COMISIÓN:
YADRAN ETEROVIC SOLANO
BEATRIZ MARÍN CAMPUSANO
BERNHARD RUMPE

Este trabajo ha sido financiada por la beca CONICYT-PFCHA/Doctorado Nacional para Extranjeros/2013-63130130, y apoyada parcialmente por los proyectos FONDEF D09I-1171 (ADAPTE) y FONDEF IDeA IT13I20010 (GEMS), y el Programa de Becas de NIC Chile.

SANTIAGO DE CHILE
2018

Resumen

La adaptación de procesos de software es la actividad de adaptar el proceso de software de una organización a las necesidades de proyectos particulares. La ingeniería basada en modelos (MDE) se ha aplicado con este fin, utilizando modelos para formalizar el proceso de software y el contexto del proyecto, y transformaciones del modelo para adaptar estos procesos. A pesar de que la adaptación basada en MDE ha demostrado ser técnicamente factible, su uso en la práctica requiere conocimiento sobre cómo adaptar procesos y construir modelos y transformaciones.

Existen algunas propuestas para la generación automática de transformaciones como una forma de reducir la complejidad de adaptar los procesos de software. Estas propuestas generalmente generan transformación solo parcialmente, y luego deben completarse manualmente. Estos enfoques no son adecuados para la adaptación de procesos de software porque no superan por completo las dificultades técnicas de adopción.

Para enfrentar estos desafíos esta tesis propone un enfoque automático de generación de transformaciones, que aborda tanto la formalidad requerida por MDE, como la usabilidad que necesitan los ingenieros de proceso a cargo de las adaptaciones. Para ello, especificamos las reglas de adaptación utilizando un lenguaje específico de dominio (DSL). Además, definimos una transformación de orden superior (HOT) que toma las reglas de adaptación especificadas como entrada y genera automáticamente la transformación de adaptación de procesos requerida. Tanto el DSL como el HOT son genéricos y, por lo tanto, pueden reutilizarse en cualquier organización. Con el fin de mejorar la usabilidad, desarrollamos un conjunto de herramientas integradas (ATAGeTT) que incorpora ambas contribuciones.

ATAGETT se aplicó en un estudio de caso exploratorio en dos pequeñas empresas de software, para evaluar su capacidad y corrección de adaptar el proceso de estas compañías. Los resultados obtenidos muestran queusuarios pudieron especificar todas las reglas de adaptación requeridas.

Luego, se llevó cabo un caso de estudio en otra empresa para validar la usabilidad de ATAGeTT y la expresividad del lenguaje de decisión propuesto. Los usuarios pudieron especificar todas las reglas de ajuste de una manera simple, y de ejecutar la adaptación de procesos de manera automática. Los resultados muestran que ATAGeTT es fácil de aprender, usable y útil para sus potenciales usuarios. Aunque los resultados aún no son suficientes, son altamente positivos y consistentes; por lo tanto, esperamos que esta propuesta pueda ayudar a mejorar esta actividad, particularmente en organizaciones pequeñas y medianas, que generalmente están más limitadas para realizar adaptacion de procesos de software.

Abstract

Software processes tailoring is the activity of adapting the organizational software process to the needs of particular projects. Model-driven engineering (MDE) has been applied with this purpose, using models for formalizing the software process and the project context, and model transformations for tailoring these processes. Even though the MDE-based tailoring strategy has proved to be technically feasible, its usage in practice requires knowledge about how to adapt the software process and how to build models and transformations.

There are some proposals for automatically generating transformations as a way to reduce the complexity of tailoring the organizational software processes. These proposals usually generate transformation only partially, and then they need to be completed manually. These approaches are not suitable in software process tailoring because they do not completely overcome the technical adoption difficulties.

Trying to deal with these challenges, we propose a complete automatic transformation generation approach that addresses both, the formality required by MDE and the usability needed by the software process engineers. To that end, we specified tailoring rules using a domain-specific language (DSL). Moreover, we define a a higher-order transformation (HOT) that takes the specified tailoring rules as input and automatically generates the required process tailoring transformation. Both, the DSL and the HOT are generic, and therefore they can be reused across organizations. In order to improve the usability, we developed an integrated tool called A Tool-set for Automatically Generating Tailoring Transformations (ATAGeTT) that incorporates both contributions.

ATAGeTT has been applied in an exploratory case study in two small software companies for evaluating its capability and correctness of tailoring the organizational software process of these companies. The obtained results show the software process engineers were able to specify all tailoring rules that were required in both companies using ATAGeTT.

Then, we conducted an explanatory case study in other software company for validating usability of the ATAGeTT and expressiveness of the proposed decision language. The software process engineers were able to specify all tailoring rules in simple manner, and the project manager participating in this study was able to execute software process tailoring in automatic way. The obtained results show the ATAGeTT is easy to learn, usable and useful for potential users. Although the results are still not enough the make strong conclusions, they are highly positive and consistent; therefore, we expect this proposal can help improve this activity in many software companies, particularly in small and medium-sized organizations that are usually more limited to perform software process tailoring.

Dedicated to my parents and brother.

Dedicated to memory of my grandfather Gregorio Silvestre, grandmother Flora Conde and great-grandmother Nicolasa Rojas.

Acknowledgments

Throughout my PhD studies, the Computer Science Department of University of Chile has showed me what it means to be a researcher and many people have helped me in different ways in the writing of this thesis. Without their support and contributions, this thesis would not exist.

First of all, I would like to thank to my parents Jorge Silvestre and Seferina Quiroga for all that they give me in my life, my brother Diego Silvestre for pushing me to follow my dreams and Mariela Pardo for her patience, ears and encouragement. And also my gratitude to my grandfather Marcos Quiroga, grandmother Marcelina Rojas, uncles, aunts, cousins for their support from Bolivia.

My special thanks to my supervisors Sergio F. Ochoa and Maria Cecilia Bastarrica for supporting and encouraging me throughout the long process of this thesis work and for their guidance and positive attitude in guiding me. I do not have words for expressing my gratitude for your human values, conversations about life, talks about all the research lines and patience with my special way of "to do science". I would like also to express my profound appreciation to Jocelyn Simmonds, Daniel Perovich and others research group members who given their direct or indirect supports in this long research journey.

It is impossible for me to write an acknowledgment letter without giving thank to thank Angelica Aguirre, Sandra Gaez and Francia Ormeño, who have been of great help in the department all these years. Of course, I want to thank all my classmates of my PhD. program. Thanks guys for every moment shared together, especially for my three closest colleagues of work: Alcides Quispe, Maira Marques and Fabian Rojas. Thank you for pushing me the many times I needed a bit of encouragement to continue.

Finally, I also recognize, with gratitude, the support from the PhD. Scholarship Program of Conicyt-Chile (Comisión Nacional de Investigación Científica y Tecnológica) (CONICYT-PCHA/2013-63130130), the NIC Chile, the University of Chile, the two Chilean government projects FONDEF D09I-1171 (ADAPTE project) and FONDEF IDeA IT13I20010 (GEMS project) for funding me several travels to conferences and helping me with additional scholarships in the last part of my stay in the PhD. program.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Software companies define their processes in order to manage their development projects in a systematic way, being able to plan, assign resources and control the progress of the project. Software processes are defined from simple elements called *process elements*. Process element names depend on the vocabulary used to define a software process; for example *role* can be used to represent somebody who performs a *task* or *activity*, or one that can generate an *artifact* or document of the software process. Formally defining the software process allows companies to rigorously document their process and tool support for formal process analysis and management, as well as making process evolution easier.

Software process modeling refers to the definition of the processes as models, plus any optional automated support available for modeling and executing the models during the software process. A software process model is an abstract representation of a software process from some particular perspective. It provides definitions of the software process to be used, instantiated, enacted, or executed. Therefore, a software process model can be analyzed, validated, simulated or executed if it is appropriately defined according to these goals. Finkelstein et al. [42] define a process model as the description of a process expressed in a suitable process modeling language. To this end, the Object Management Group (OMG)[1] proposed the Software and Systems Process Engineering Metamodel specification 2.0 (SPEM 2.0) [55] that is a standard for modeling software processes.

Development projects faced by a particular company may be of different kinds, e.g., large or small, complex or simple, new development or evolution, and therefore the same software process is not equally appropriate for addressing all of them. Software process tailoring is the activity of adapting a general software process to match the needs of the project at hand according to its particular characteristics. This activity is generally performed together with the process engineer and the project manager. The *process engineer* knows about the organizational process and how the project characteristics affect tailoring, while the *project manager* knows about the project's particular characteristics. There are several strategies for software process tailoring, including template-based tailoring [29], framework-based tailoring [15], constructive tailoring [126] and automated tailoring [65]. Empirical studies show that process

[1]OMG website: http://www.omg.org/spec/SPEM/2.0/

tailoring is difficult because it involves intensive knowledge generation and application [27].

Model-driven Engineering (MDE) [83] is a software development paradigm that can help address this difficulty. MDE promotes the use of models, as the main and first class elements to face the difficulties of third generation programming languages for dealing with platform complexity and for efficiently expressing domain-specific concepts. The MDE paradigm advocates the use of models as the most important artifacts in the software development process, while artifacts such as documentation and source code can quickly be produced from those models by using automated transformations [45].

A model transformation takes one or more source models as input and produces one or more target models as output, following a set of transformation rules [139]. In this way, MDE can be applied to tailor process models. MDE-based software process tailoring proposed in [66] requires a definition of the project context and the general software process along with its variability, i.e., the process elements that might be included or not during process tailoring. These models are used as input of a transformation whose output is the adapted software process model. Although this proposal reaches the expected technical results, there are still usability factors that make it difficult to deploy it in software industry [78]. First, current transformation languages and tools are not simple for defining and applying tailoring transformations. Second, the potential users -process engineers and project managers- do not usually have the required knowledge about MDE.

As a means to address these challenges, this thesis proposes the automatic generation and execution of tailoring transformations through the interactive definition of transformation rules, thus improving the usability and hiding the complexity of MDE components. The transformation rules are defined using a high-level language and they are formally specified as a model that we call a *variation decision model*. A higher-order transformation takes this variation decision model as input and generates a transformation in an automatic way. This resulting transformation can then be used for software process tailoring. In order to improve the usability of the proposed solution, we developed an integrated tool as front-end for defining transformation rules and generating tailoring transformations without interacting with MDE components. The back-end of this integrated tool is implemented as a megamodel that can support process modeling, tailoring and evolution.

The following sections describe the motivation and the problem addressed. We then present the general research question, hypotheses, goals, research method and the outline of this thesis document.

1.1 Motivation

A software process is a structured set of activities required to develop a software system [95]. There are several kinds of software processes from traditional software development processes, such as the Waterfall [131], to more modern ones such as Rational Unified Process (RUP) [88], Scrum [136] or Extreme programming (XP) [9]. Having a defined process allows companies to analyze a project's results in terms of quality and productivity as a basis

for improvement. Moreover, if the process is rigorously defined, the company can get an ISO 12207 certification [150] or a CMMI evaluation [154] that may give them a commercial advantage.

Software companies get involved in different kinds of projects; e.g., large software developed from scratch, developments with an experienced or a novice team, bug fixing, and software projects using well–known or highly innovative technologies. Therefore, the same process is not usually appropriate for addressing all kinds of projects, or it is not equally productive and effective in all cases. If the defined software process is not easily understood and applied, investing time and money on its definition is not productive for software companies. Therefore, an obvious conclusion is that it is necessary to count on a series of processes, each one for each kind of project, i.e., a process family. Evolving several processes independently implies significant effort, as well as a risk of introducing inconsistencies among these processes.

In this scenario, Software Process Lines (SPrL) appear as a potential approach to deal with this challenge. A SPrL is a software product line where the products are software processes [111]. In this way, a SPrL defines its reusable process assets and a mechanism for obtaining particular processes within the SPrL scope; this mechanism corresponds to process tailoring. MDE techniques can contribute to developing SPrL.

An MDE-based SPrL was presented by Hurtado et al. [65, 66], in which two source models were considered as input for a process tailoring transformation. The transformation for tailoring the process model was called *tailoring transformation* and it was implemented using the Atlas Transformation Language (ATL) [76]. The tailoring transformation generates an *adapted software process* model that includes only the activities that are required for addressing projects in a particular context and nothing else. The input models used by such a transformation are: (1) the *software process model* of a certain company based on SPEM [55], and (2) the *context model* for a project where the process needs to be applied. Figure 1.1 shows the structure of the proposal.

This MDE-based tailoring approach obtains the adapted software process model, but it requires defining the software process model that conforms to the SPEM metamodel, identifying the process potential variability, the project context model and its corresponding metamodel, as well as programming the tailoring transformation itself.

MDE solutions are powerful and suitable for different application domains, but they are almost always complex. There are social and organizational factors that threaten their industrial adoption [67] (e.g., technological factors, and resistance to change). Probably, if we could lower complexity, then adoption of this solution would be higher. In this sense, if software companies want to reuse their defined processes through model-based tailoring by defining and applying the transformation rules, they should not require direct interaction with the source code of any model transformation. This is particularly important for small software companies, which usually are fragile in terms of human, technological and economic resources [68, 157].

Moreover, writing tailoring transformations requires not only knowledge about transformation programming languages, but also about the particular process model and the way

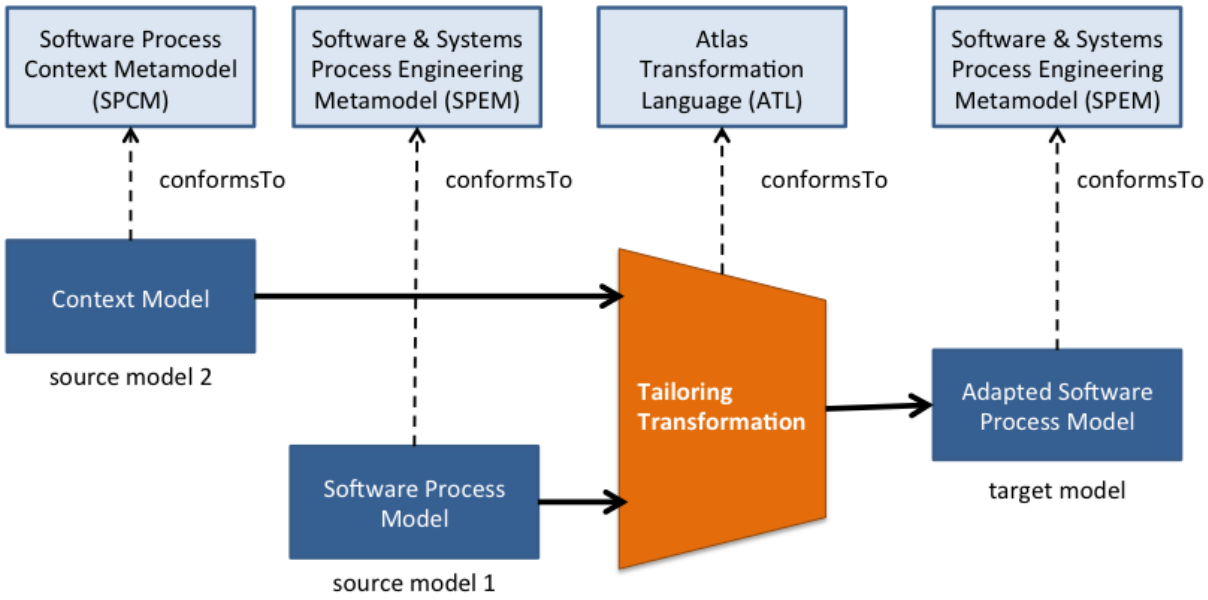


Figure 1.1: MDE-based strategy for tailoring software process [65].

its variability should be resolved according to the project context characteristics. These two kinds on knowledge –programming a model transformation and configuring a software process– are almost never mastered by the same person in the company, and it is even less frequent in small and medium-sized software companies, where the average expertise of the staff is not high [125]. Even if someone possessed both kinds of knowledge, manually writing and maintaining complex model transformations would still be a time-consuming and error-prone activity. Moreover, the whole effort invested in this endeavor cannot be reused across different companies.

1.2 Problem Statement

MDE-based approaches in general allow software modeling at different abstraction levels and addressing different application domains [86]. However, they require mastering complex concepts and formalisms relating model definition and writing model transformations in specific languages. Particularly, generating appropriate tailoring transformations requires two different kinds of knowledge: (1) how the process should be tailored and (2) how to build models and transformations.

On the one hand, process engineers -at least in Chile- are in charge of tailoring the organizational software process and should know precisely how the context attribute values impact the process variation, but they are almost never experienced in the use of transformation languages and MDE concepts. On the other hand, project managers know about the characteristics of the project to address and so they can define the project context for software process tailoring, but they do not know how to execute the tailoring transformation. MDE knowledge can be obtained through training in both cases, but highly skilled people

are still required, and these people are not generally in charge of the software process in small and medium-sized software companies. In this scenario, generating transformations automatically seems an appealing solution.

There are some proposals for partially generating model transformations, but they are not easily applicable in the software industry, because there are still several factors that jeopardize their usage in software companies, such as:

- Model transformation development is expert-dependent in MDE. Moreover, it is not enough to hire a specialized professional to define these model elements, since all of them evolve: variable process elements, context attributes and potential values.
- The process tailoring knowledge encapsulated within the transformation rules is generally only managed by the company's process engineer, but he/she is usually unfamiliar with the syntax and semantics of transformation languages.
- The current transformation languages and tools are not simple for defining and applying tailoring transformations, even for trained engineers.

1.3 Research Question and Hypotheses

1.3.1 General research question

Considering the problem stated in Sec. 1.2, this thesis addresses the following general research question: *Can software process tailoring be successfully conducted in an expert-independent way?*. This question intends to explore if software process tailoring can be conducted in an easy and successful way by process engineers and project managers that do not necessarily have knowledge in MDE. The usability of software process tailoring should consider the potential capabilities of these real users. In order to improve the adoption of MDE-based tailoring approaches, the software process tailoring solution should be thus expert-independent and automatic.

1.3.2 Hypotheses

By considering the stated problem and research question, the hypotheses of this thesis work are the following:

- **H1:** The complexity of using the MDE-based tailoring approach can be hidden through the use of a high-level language that allows defining tailoring rules using just processes concepts and project characteristics.
- **H2:** The MDE-based tailoring approach allows automatic generation and execution of the tailoring transformations that software companies may require.
- **H3:** The whole MDE-based tailoring approach can be implemented as a usable organization-independent tool.

1.4 Research Goals

The main goal of this thesis is to define a generic and usable tool for generating and executing process tailoring transformations. This work intends to improve the usability of the current solutions for transformation generation through a graphical environment that includes only domain concepts. It also achieves the required expressiveness for the generated tailoring rules.

Based on the main goal, the specific goals of this thesis are the following:

- **G1:** Define a high-level language for specifying tailoring rules.
- **G2:** Specify and develop a usable graphical environment for defining and executing tailoring rules.
- **G3:** Generate tailoring transformations in an automatic way by using tailoring rules as input.

1.5 Research Methodology

The research method used in this thesis is based on controlled experiments and case studies. Typically, controlled experiments involve a set of experiments for formulating hypothesis by looking for changes brought on by alterations to a particular variable [147]. Thus, controlled experiments allow us to focus on observing specific variables and relationships between them.

On the other hand, a case study is an empirical inquiry that investigates a contemporary phenomenon within its real-life context [173]. According to Runeson and Höst [133], a case study is an appropriate method for this type of research, because in other cases it would be difficult to control all of the experimentation variables and understand the software process tailoring and process analysis phenomena in industrial settings. Case studies use a variety of evidence from different sources [127], such as documents, artifacts, interviews and observation, surpassing the range of sources of evidence that might be available in a historical study.

The research methodology used in this thesis has four phases: (1) experimentation, (2) exploration, (3) formulation and (4) validation. Figure 1.2 shows methodology phases and their associated research methods.

A controlled experiment is defined for evaluating the *feasibility* of transformation rules definition and tailoring transformation generation. The controlled experiments consider a real *Software Process* that is used for generating another *Experimental Software Process*. The transformation rules definition and tailoring transformation generation are applied using a *Proof-of-Concept* that is a basic interface (see Fig. 1.3). In this case, the controlled experiments have three objectives: (1) characterize the transformation rules, (2) develop a basic interface for defining transformation rules in an interactive way, and (3) using generative programming for obtaining the tailoring transformations [31]. Finally, the controlled

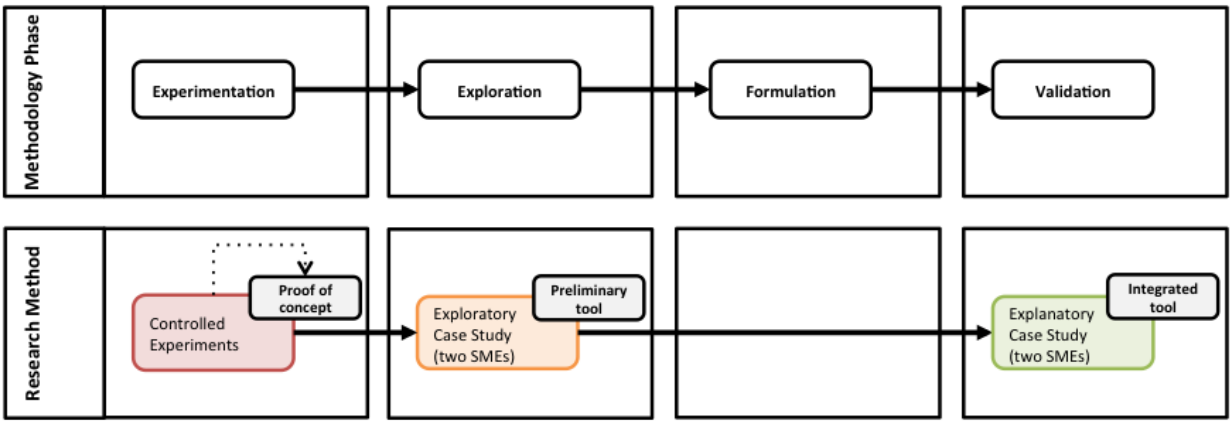


Figure 1.2: Phases of the research methodology.

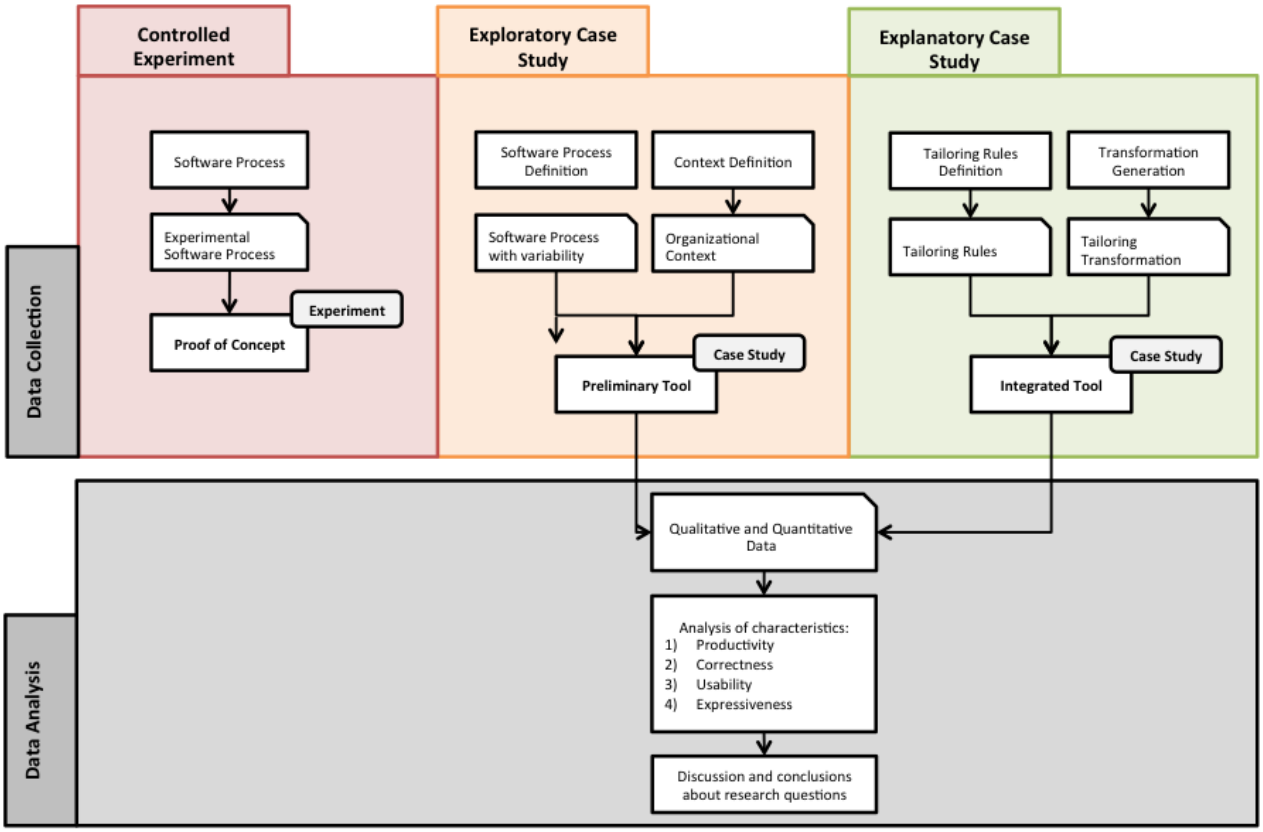


Figure 1.3: Detailed structure of the research methodology.

experiment is evaluated using inspection of the transformation rules and testing the generated transformation.

An exploratory case study is applied for evaluating *correctness* and *productivity* of the proposed solution in two small and medium-sized Chilean software companies: Mobius[2] and Rhiscom[3] respectively. The exploratory case study is used for evaluating syntactic correct-

[2]Mobius website: http://www.mobius.cl
[3]Rhiscom website: http://www.rhiscom.com

ness (well-formed output model) and productivity (identification of missing and extra tasks) of two preliminary tool components: (1) transformation rules definition and (2) tailoring transformation generation. This case study considers the definition of a *Software Process with variability*, and *Context Definition* for generating a *Organizational Context*. The proposed solution includes a *Preliminary Tool* that automatically generates transformations (see Fig. 1.3).

An explanatory case study is applied for evaluating the integrated tool (in terms of *usability* and *expressiveness* of the proposed solution) in one medium-sized Chilean software company: Ki teknology[4]. The evaluation of expressiveness of the high-level language consists in verifying that it counts on all the required constructs for expressing the software process engineer's intentions for tailoring software process. The evaluation of usability of the integrated tool consists in applying structured questionnaires to process engineers about their perception and potential adoption of the tool for three components: (1) transformation rules definition, (2) tailoring transformation generation and (3) software process tailoring execution.

The evaluation of the proposed solution was conducted in various small and medium-sized Chilean software companies that were part of the Adaptable Domain and Process Technology Engineering (ADAPTE) research project[5]; and the Experimental Management of Software Improvement (GEMS) project[6]. In the ADAPTE project, the organizational software processes of various companies were formalized using the Eclipse Process Framework (EPF) [7]. In the GEMS project, the use in practice of these software processes were analyzed in terms of software process components (tasks and work products), and the software process variability was also specified in the EPF. The software process variability considers: optional elements, and alternative elements. At tailoring time, the optional elements may be removed from organizational software process and the alternative elements are chosen from a given set.

1.6 Contributions

The main contributions of this thesis are depicted in Fig. 1.4 and can be grouped in two areas:

- Scientific Contributions:
  - A domain-specific language (DSL) that supports the definition of tailoring rules. The DSL allows the software process engineer to define tailoring rules in a simple way. We call this DSL Variation Decision Model (VDM). The Variation Decision Metamodel was thus specified for defining the abstract syntax of the language, describing the elements provided by the language and how they may be combined to instantiate models. Transformation rules are specified in a model that conforms

[4]Ki Teknology website: http://www.kiteknology.com
[5]ADAPTE website: http://www.adapte.cl
[6]GEMS website: http://www.dcc.uchile.cl/gems
[7]EPF website: http://www.eclipse.org/epf

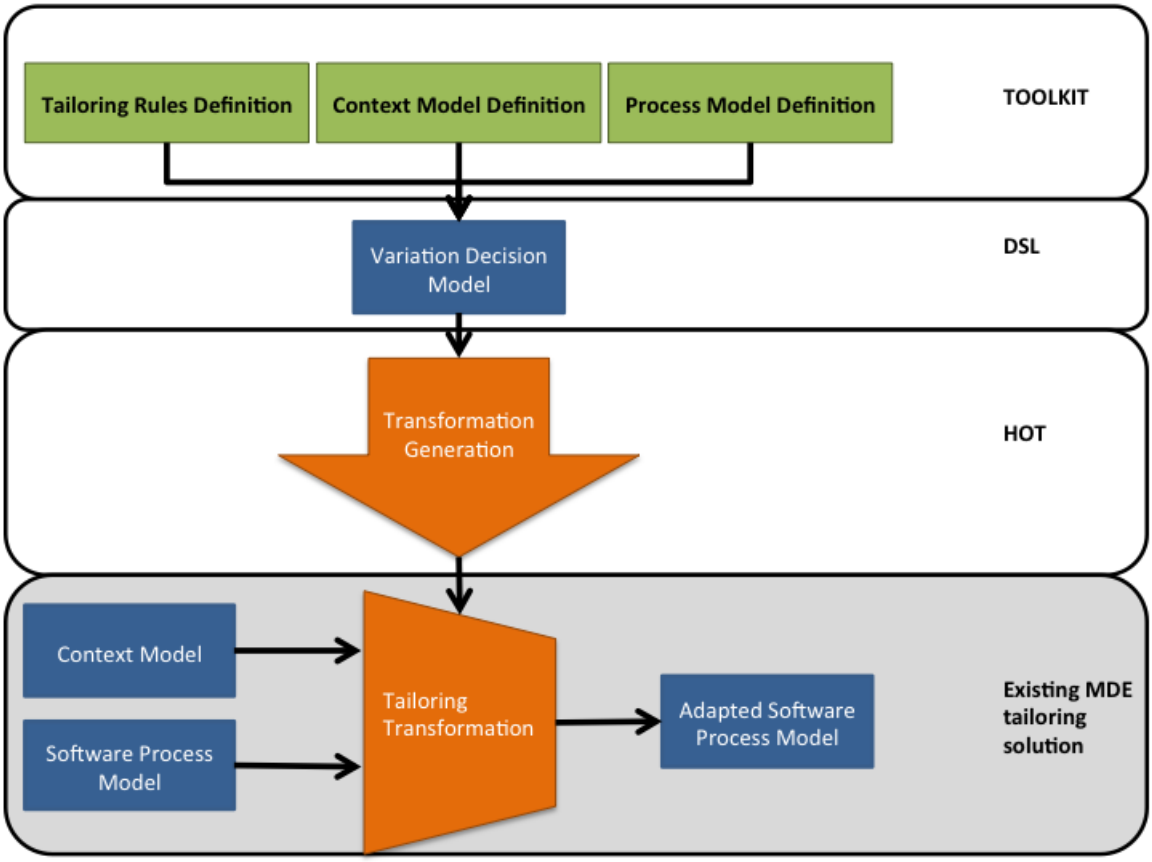


Figure 1.4: Contributions of this Thesis.

to this metamodel.

The DSL reduces the complexity of defining tailoring rules by just using domain concepts: process and context elements.

- A higher-order transformation (HOT) that takes the variation decision model that specifies the tailoring rules and automatically generates the tailoring transformation. This is achieved in two steps: (1) an M2M transformation takes the VDM as input and generates the tailoring transformation model, and (2) a M2T transformation takes the tailoring transformation model as input and generates the tailoring transformation code, that is finally what needs to be executed. This mechanism allows users to generate tailoring transformations in an automatic way without interacting directly with models and transformations.

It is worth noting that both, the DSL and the HOT are organization independent, and therefore they can be reused across different organizations.

- Industrial Contributions:
  - A graphical environment for specifying tailoring rules. This environment allows software process engineers to define tailoring rules for each variation point of the software process according to the values in the organization context model.
  - A usable integrated tool for adapting software processes according to project contexts. The integrated tool allows for the generation and execution of tailoring transformations in an automatic way hiding the complexity of managing models

and writing model transformations.

This tool-set is also organization independent.

1.7 List of Publications

During the development of this thesis, the author published in the following papers:

International Conferences and Workshops

- **Luis Silvestre**, María Cecilia Bastarrica, Sergio F. Ochoa. "Reducing Complexity of Process Tailoring Transformations Generation", Model-Driven Engineering and Software Development, Second International Conference, MODELSWARD 2014, Lisbon, Portugal, January 7-9, 2014, Revised Selected Papers, Springer International Publishing, vol. 506, pp. 171–182, December, 2015.
- **Luis Silvestre**. "Automatic Generation of Transformations for Software Process Tailoring", ACM Student Research Competition at 18th International Conference on Model Driven Engineering Languages and Systems, MoDELS'2015, CEUR-WS, vol. 1503, pp. 46–51, Ottawa, Canada, September 30, 2015.
- Jocelyn Simmonds, Daniel Perovich, María Cecilia Bastarrica, **Luis Silvestre**. "A megamodel for Software Process Line modeling and evolution", 18th International Conference on Model Driven Engineering Languages and Systems, MoDELS'2015, IEEE, pp. 406–415, Ottawa, Canada, September 30, 2015.
- **Luis Silvestre**, María Cecilia Bastarrica, Sergio F. Ochoa. "A Usable MDE-based Tool for Software Process Tailoring ", Posters and Demos at 18th International Conference on Model Driven Engineering Languages and Systems, MoDELS'2015, CEUR-WS, vol. 1554, pp. 36–39, Ottawa, Canada, September 27, 2015.
- Felipe González, **Luis Silvestre** and María Cecilia Bastarrica. "Template-Based vs. Automatic Process Tailoring", In proceedings of the XXXIII International Conference of the SCCC, pp. 124-127, Talca, Chile, November 2014.
- **Luis Silvestre**. "A Domain Specific Transformation Language to Support the Interactive Definition of Model Transformation Rules", Doctoral Symposium at 17th International Conference on Model Driven Engineering Languages and Systems, MoDELS'2014, CEUR-WS, vol. 1321, Valencia, Spain, September 30, 2014.
- María Cecilia Bastarrica, Gerardo Maturro, Romain Robbes, **Luis Silvestre**, René Vidal. "How does Quality of Formalized Software Processes Affect Adoption?", 26th International Conference on Advanced Information Systems Engineering, CAiSE'2014, LNCS 8484, pp. 226–240. Springer International Publishing Switzerland, Thessaloniki, Greece, 16-20, June 2014.
- María Cecilia Bastarrica, Jocelyn Simmonds, **Luis Silvestre**. "Using Megamodeling to Improve Industrial Adoption of Complex MDE Solutions", 6th Workshop on Modeling in Software Engineering, MiSE'2014, Hyderabad, India, pp. 31-36, May 31th - June 7th, 2014.
- **Luis Silvestre**, María Cecilia Bastarrica and Sergio F. Ochoa. "A Model-Based Tool for Generating Software Process Model Tailoring Transformations". 2nd International

Conference on Model-Driven Engineering and Software Development, MODELSWARD 2014. Lisbon, Portugal, pp. 533-540, January, 2014.

- **Luis Silvestre**, María Cecilia Bastarrica and Sergio F. Ochoa. "Implementing HOTs that Generate Transformations with Two Input Models", In proceedings of the XXXII International Conference of the SCCC, pp. 26–29, Temuco, Chile, November, 2013.
- Jocelyn Simmonds, María Cecilia Bastarrica, **Luis Silvestre**, Alcides Quispe. "Variability in Software Process Models: Requirements for Adoption in Industrial Settings", 4th Workshop on Product LinE Approaches in Software Engineering (PLEASE 2013) in 35rd International Conference on Software Engineering (ICSE 2013), pp. 33–36, San Francisco, USA, May, 2013.
- Daniel Ortega, **Luis Silvestre**, María Cecilia Bastarrica, Sergio Ochoa. "A Tool for Modeling Software Development Contexts", In Proceedings of the XXXI International Conference of the SCCC, pp. 29–35, Valparaíso, Chile, November, 2012.
- Aldo Bertero, **Luis Silvestre**, María Cecilia Bastarrica. "Text-to-Model and Model-to-Text Transformations between Software Processes and Software Process Models", In Proceedings of the XXXI International Conference of the SCCC, pp. 36–40, Valparaíso, Chile, November, 2012.

Technical Reports

- TR/DCC-2015-1 Jocelyn Simmonds, Daniel Perovich, Cecilia Bastarrica and **Luis Silvestre**. "Software Process Line Modeling and Evolution", May 6, 2015.
- TR/DCC-2014-1 María Cecilia Bastarrica, Jocelyn Simmonds, **Luis Silvestre**. "A Megamodel for Process Tailoring and Evolution", January 20, 2014.
- TR/DCC-2013-7 **Luis Silvestre**, María Cecilia Bastarrica and Sergio F. Ochoa. "A Model-Based Tool for Generating Software Process Model Tailoring Transformations", September 04, 2013.
- TR/DCC-2012-3 Jocelyn Simmonds, María Cecilia Bastarrica, **Luis Silvestre** and Alcides Quispe. "Modeling Variability in Software Process Models", March 5, 2012.
- TR/DCC-2011-12 Jocelyn Simmonds, María Cecilia Bastarrica, **Luis Silvestre** and Alcides Quispe. "Analyzing Methodologies and Tools for Specifying Variability in Software Processes", November 4, 2011.

1.8 Outline of the Thesis

The thesis is structured as follows:

Chapter 2 provides the background about basic concepts used in this thesis.

Chapter 3 describes the previous research about MDE-based tailoring approach. This chapter also identifies initial problems to be addressed and presents the previous work about tools for defining contexts and generating software process models.

Chapter 4 presents the domain-specific language for formally defining tailoring rules. To

this end both, the abstract and the concrete syntax are presented.

Chapter 5 presents the tailoring transformation generation using a higher-order transformation. The higher-order transformation takes the tailoring rules definition and automatically generates the tailoring transformation.

Chapter 6 presents the integrated tool for automatically generating tailoring transformations and executing software process tailoring. The user interface and all components are described as well as a running example that illustrates its usage.

Chapter 7 presents an exploratory case study for evaluating correctness and productivity of the integrated tool in one small and one medium-sized Chilean software companies.

Chapter 8 presents an explanatory case study that analyzes and discusses the expressiveness and usability of the integrated tool when applied in other medium-sized Chilean software company.

Finally, **Chapter 9** presents conclusions, contributions and discusses limitations of the proposed solution.

Chapter 2

Basic Concepts

This chapter introduces basic concepts and definitions in the field of software processes and model-driven engineering. Section 2.1 explains basic concepts about software process, software process modeling and software process lines. Section 2.2 addresses model-driven engineering, models, metamodels, metametamodels and model transformations.

2.1 Software Process

In software engineering, various definitions of software processes can be found. Although differences of opinion still exist, there appears to be some agreement that a software process refers to a set of organizational activities (sometimes as a workflow) for managing development projects in a systematic way, as stated by Humphrey [60] and Somerville [148]. In addition, Feiler and Humphrey [40] consider a software process as a set or sequence of steps followed *to reach a defined goal*, whereas Fuggetta et al. [46] claim that a software process is *a set of patterns or activities* compiled to find a solution to a series of software development problems. ISO/IEC 12207 [150] defines the software development process as a set of interrelated or interacting activities, which *transforms inputs into outputs*, where the standard embodies the process as an outcome-based instrument that *should be beneficial to the stakeholders*. Finally, Humphrey [61] says that a software process is *a set of tools, methods, and practices* for producing software products. Taking into consideration all of these features presented by different authors, we consider the following definition.

Definition 2.1 *A software process is a set of tools, methods, practices, patterns, and activities organized that transform inputs into outputs so that a goal may be reached in order to serve the interest of its stakeholders.*

The goal of a software process is to provide a roadmap for the production of high quality software products that meet the needs of their stakeholders within a balanced schedule and budget [174].

If a company aims to certify or evaluate its software process, it should be strictly defined with some model and standard such as CMMI [154] or ISO/IEC 12207 [150]. Formally defining the software process allows software organizations to use supporting tools for formal process analysis and management, also facilitating evolution of software processes. Software processes can evolve in different ways, such as adding or removing activities, changing a work product template, or assigning a new role to a particular task, among others.

2.1.1 Software process modeling

A software process can be expressed in an *abstract representation* as a model. A software process model is a description of a software process *from some particular perspective* and it is used to *analyze, understand and improve* the specific process [2]. According to Selby [137], a software process model is used to refer to the development activities and the ways to control the software product by understanding the steps taken throughout a process. Finkelstein et al. [42] define a software process model as the description of a process *expressed in a suitable process modeling language*. Considering these definitions we will consider an extension of that proposed by Lonchamp [95].

Definition 2.2 *A software process model is an abstract description of a software process from a particular point of view, using a suitable process modeling language for representing, tailoring, enacting, simulating or executing a software process.*

Process engineers are usually responsible for developing flexible and customizable software processes. To support authoring and customization of software processes, the Object Management Group (OMG)[1] proposes the Software and Systems Process Engineering Metamodel (SPEM 2.0) specification [55]. It is an industry open standard for modeling software processes.

In SPEM, a software process is a collection of activities where an activity is a "big-step" that groups roles, work products and task uses. Roles perform activity tasks, and work products usually represent the input/output artifacts for tasks. Software process elements (like tasks, roles and work products) are defined and stored in a method library. These definitions can later be reused in multiple software process definitions.

SPEM provides a standardized and managed representation of method libraries in order to allow reuse of methods content. It aims to support development practitioners in defining a knowledge base for software development. It allows them to manage and deploy method library content using a standardized format. SPEM also provides a standardized representation of process structure for defining an independent life cycle, allowing method content to be placed into a specific process life cycle. Such processes can be represented as workflows and/or breakdown structures.

Enterprise Architect (EA)[2], IBM Rational Method Composer (RMC)[3] and Eclipse Process

[1]OMG website: http://www.omg.org/spec/SPEM/2.0/
[2]EA website: http://www.sparxsystems.com.au/products/ea/
[3]RMC website: http://www-03.ibm.com/software/products/en/rmc

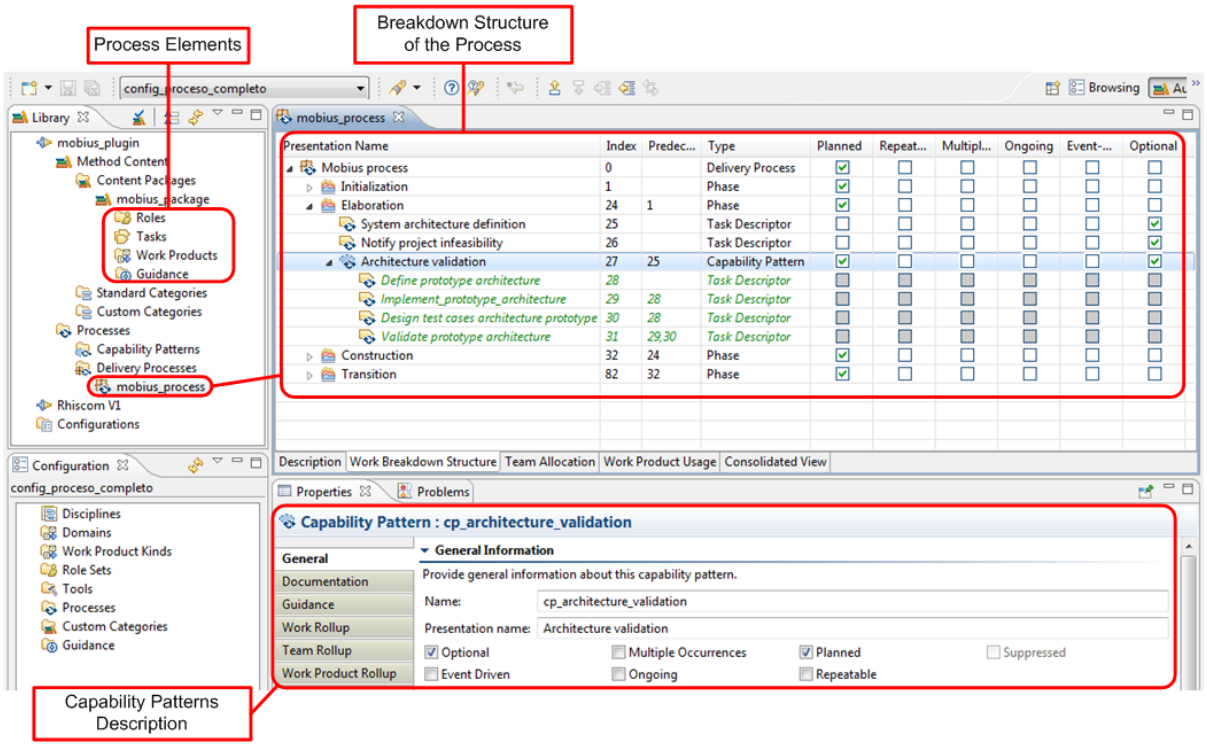


Figure 2.1: Example of software process breakdown using EPF.

Framework (EPF)[4] are three popular integrated development environments (IDE) used to specify software processes in SPEM. Figure 2.1 shows a software process breakdown that describes all SPEM's software process elements (e.g., phases, capability patterns, activities, tasks, roles, and work products) along with its relationships using EPF.

In EPF, the process elements are defined in the *Method Content* and can be categorized as roles, tasks, work products and guidance. The process elements are the building blocks from which software processes are composed as breakdown structure. The process elements used in the breakdown structure can have a description about its complementary information as: name, presentation name and its software process characteristics. Therefore, the breakdown structure in EPF supports bridging the gap between process elements definition and process enactment in projects (see Fig. 2.1).

The EPF allows visualizing software process behavior using activity diagrams, even though this is not part of SPEM. Figure 2.2 shows an activity diagram for the *Architecture validation* capability pattern that is part of the software process breakdown shown in Fig 2.1. Moreover, EPF allows visualizing other diagrams that are associated with the activity diagram. Figure 2.3 shows a diagram based on the *Developer* role that is in charge of the set of tasks, which takes work products as input and/or output.

A well-defined software process model is a determinant factor for achieving quality products and productive projects [60]. However, defining a software process model demands an enormous effort for making explicit common practices and defining desired practices that may not yet exist within the company.

[4]EPF website: https://eclipse.org/epf/

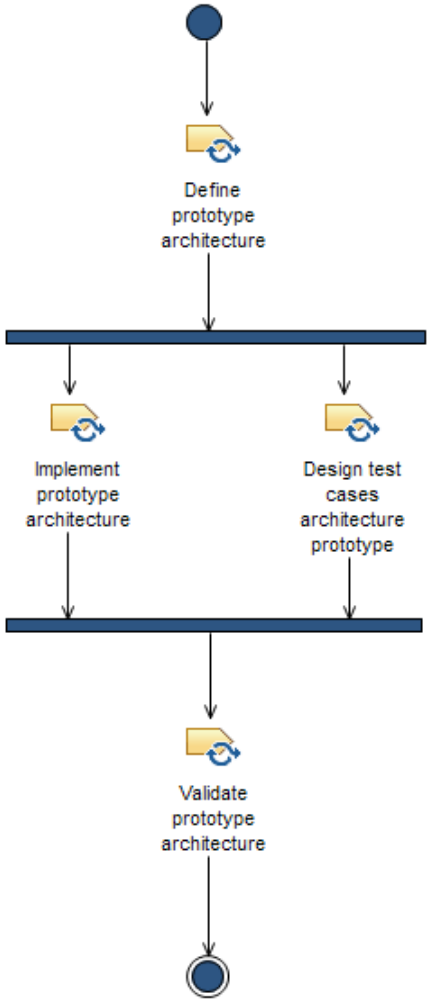


Figure 2.2: Example of activity diagram of the *Architecture validation* activity using EPF.

2.1.2 Software process lines

Software companies define software processes for planning and guiding projects. However, software process definition is expensive, and in practice, no one process fits all projects. Consequently, the formalized process should be adapted to each project context before it can be used. As mentioned before, an alternative to address this challenge is to use Software Process Lines (SPrL).

SPrL are software product lines (SPL) where the products are software development processes [111]. The SPrL approach, similar to SPL, proposes the formalization of this set of software processes as a base software process, which represents its common parts, along with its potential variability. Consequently, a SPrL is an appropriate way to define, tailor and evolve a set of related processes. This opinion is supported by the work on process variability representation [145], process families [128], SPrL architectures [167], process domain analysis [106], and SPrL scoping [3].

Table 2.1 shows the stages of a SPL according to [115] and the correspondence with each

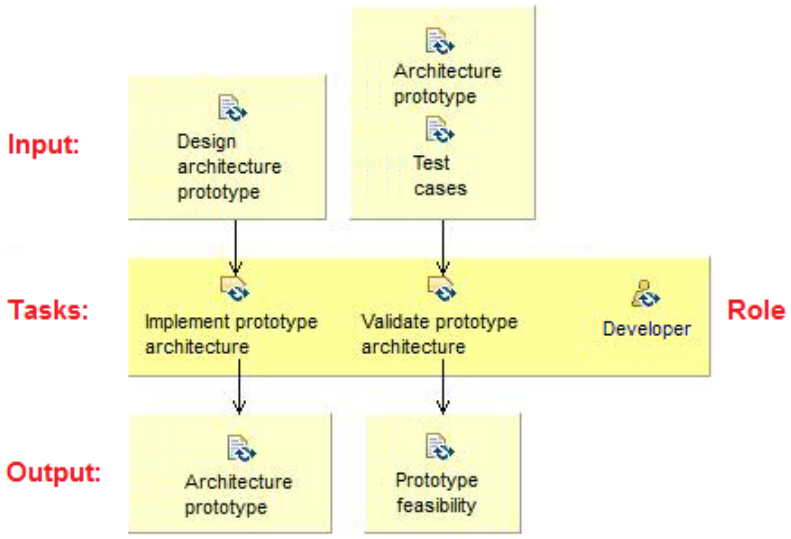


Figure 2.3: Partial view of tasks and work products related to the developer role, as part of the activity diagram shown in Fig. 2.2 using EPF.

Table 2.1: Software process lines as product lines.

Domain Modeling	**Architecture Modeling**	**Process Line Implementation**	**Process Line Testing**
• Organizational Process (including variability) • Organizational Context Definition	• Organizational Process Formalization • Organizational Context Modeling	• Method Library Definition • Organizational Process in Use	• Organizational Process in Use Testing
Product Requirements	**Product Architecture Definition**	**Product Instantiation**	**Product Testing**
• Product Context Modeling	• Process Tailoring or Process Selection	• Process Configuration	• Process Configuration Testing

phase of SPrL. The process of instantiating a SPrL to a particular project context is called process tailoring [8]. As a typical SPrL instantiation, software process tailoring is the activity in which a software process variation points are resolved, so that it is adapted to deal with the particular characteristics of a project.

SPrL has been used in software companies for generating software processes adapted to specific project contexts [63]. In this way they achieve higher productivity as no extra work is required, while also achieving higher product quality, as no required work is left out. All of this accomplished by reusing process assets. Therefore, SPrL promotes software process asset reuse, since common elements are defined only once, and specific processes are generated through software process tailoring.

Software process tailoring requires knowledge about the organizational process of the company, as well as potential project contexts. This activity must be carried out in a structured manner [30] in order to ensure that the resulting process meets the needs of the project. There are different approaches to process tailoring, such as counting on a set of predefined processes, building a software process by configuring a series of process elements, or generat-

ing particular processes by customizing a general predefined process that includes or not its potential variability. All these approaches may be supported by tools if the software process is formalized, but each one has its own challenges.

2.2 Model-driven Engineering

Software systems are reaching such a high degree of complexity that even current third-generation programming languages like Java or C#, are sometimes unable to efficiently support their creation [135]. One of the problems with these languages is that they are still too oriented toward specifying how the solution should work, instead of what the solution should be. This leads to a need for mechanisms and techniques that allow the developer to abstract over the programming language itself and focus on creating a solution for a certain problem.

The Model-driven engineering paradigm promotes the use of models to raise the level of abstraction and automation (of the model manipulation operations) in the software development [135]. Abstraction is a primary technique to deal with complexity, whereas automation is an effective method for promoting productivity and quality. The objective is to increase productivity and reduce time-to-market, by enabling the development of complex systems by means of abstract models defined with concepts that are much less bound to the underlying implementation technology and much closer to the problem domain [11].

The MDE paradigm promotes the use of models as the main and first class elements to face the difficulties of third generation programming languages for dealing with platform complexity and for efficiently expressing domain-specific concepts [83]. This paradigm includes the following concepts [135]:

Transformation Engines and Generators analyze certain aspects of the models and are able to create various kinds of artifacts, such as source code, input for simulation, use description, or alternative representations of the models.

Domain-Specific Modeling Languages (DSML) focus on modeling and are used for formalizing the application structure, behavior and requirements within particular domains.

MDE has been applied in different areas such as the development of applications [5], automatic change evolution [52] and also in software process engineering [22].

2.2.1 Technical spaces

A technical space (TS) is defined as a working context with a set of associated concepts, body of knowledge, tools, required skills, and capabilities [91]. The artifacts of which a software system is composed conform to the TS used to develop them. For example, the source code of a program conforms to the grammar TS, called *grammarware* [87], whereas XML documents conform to the XML TS, that we can call *xmlware*. Similarly, models conform

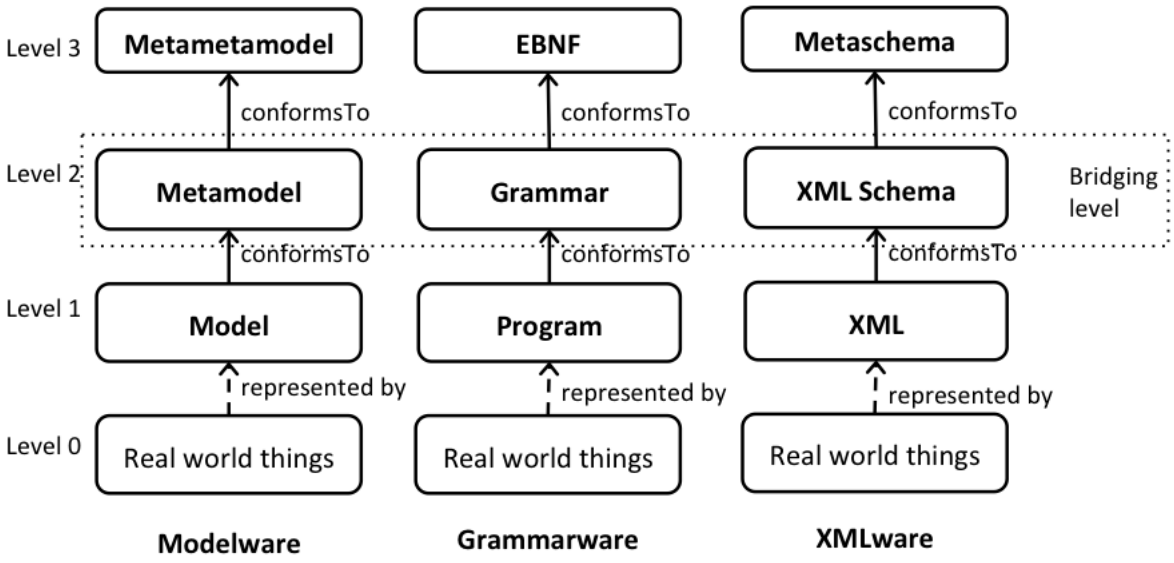


Figure 2.4: Technical spaces (adapted from Kurtev [91]).

to the MDE TS, called *modelware* [171]. A TS is defined based on a number of artifacts in different abstraction levels, which are related to each other by means of a conformance relationship (e.g., program/grammar in grammarware, xml document/schema in xmlware and model/metamodel in modelware).

A technical space is a model management framework with a set of tools that operate on the models definable within the framework [92]. However, TSs are not isolated and bridges can in fact be defined between two different TSs. Bridging TSs allows the artifacts created in one TS to be transferred to another different TS in order to use the best capabilities of each technology, promoting interoperability between applications. When bridging the MDE TS with other TSs, two operations should be supported: (1) the extraction of models from software system artifacts defined in the TS considered and (2) the generation of these artifacts from models. For instance, a bidirectional grammarware-modelware bridge should provide both a model extractor from source code and a source code generator from models.

For example, Fig. 2.4 shows a bridge between the modelware, the grammarware and xmlware technical spaces. *Model* and *Program* are two representations of the same system in those technical spaces. Bridging those two worlds enables the generation of a program from a model and vice versa. This in turn enables the reuse of existing artifacts. Moreover, artifacts can be built within the technical space where their construction and operation is more cost-effective. This implies the availability of appropriate artifacts (called extractors) that are able to extract knowledge from a TS and of others called injectors that are able to inject the knowledge into another TS.

2.2.2 Models and metamodels

As models play an important role in MDE, a step to the definition of models is to represent the models themselves as instances of some more abstract models. A model is a representation

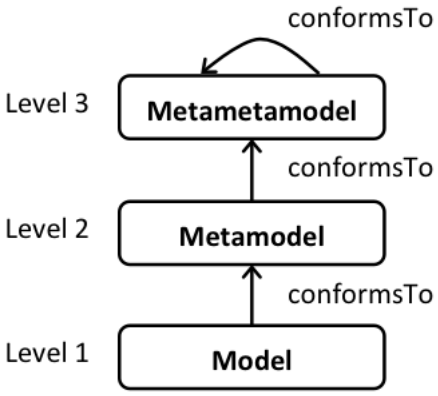


Figure 2.5: Three-level architecture in MDE (adapted from Bezivin [11]).

containing the essential structure of some object or event in the real world [135]. A model can be viewed as a reduced specification of a system that represents entities composing software artifacts/concepts in the real world [11]. The main assumption of MDE is that a model, and not the programming source code, is the right representation level for managing all artifacts within a software engineering process. Figure 2.5 shows a three-level architecture in MDE for creating models of any domain:

- **Level 3:** The metametamodeling level forms the foundation of the metamodeling hierarchy. The primary responsibility of this level is to define the language for specifying a metamodel.
- **Level 2:** The metamodeling level is an instance of a metametamodel, meaning that every element of the metamodel is an instance of the metametamodel. The primary responsibility of the metamodel level is to define a language for specifying models.
- **Level 1:** The modeling is an instance of a metamodel and it is a simplified representation of a certain reality, according to the rules of a certain language.

Models, metamodels and metametamodels are related in terms of a conformance relation. Such relation is equivalent to the relation the code written in a programming language must conform to the grammar rules of that language. Moreover, this three-level architecture may be implemented according to different modeling standards. For instance, the OMG proposes a standard metametamodel called Meta Object Facility (MOF) [56], and it is used as the metametamodel for defining the Unified Modeling Language (UML) [56] metamodel. The UML metamodel allows users to specify their own UML models.

Considering the three-level architecture in MDE and their relations, we can define the following concepts [11, 18, 138]:

Definition 2.3 *Conforms-to is a relation where a model M conforms to a metamodel MM if it is created using the modeling language which is based on MM.*

Definition 2.4 *Modeling language is any graphical or textual computer language that provisions the design and construction of structures and models following a systematic set of rules and frameworks.*

2.2.3 Model transformations

A benefit of the three-level architecture in MDE is that models can be transformed into other models. In other words, a model conforming to one metamodel can also be transformed into a model conforming to another metamodel. An important feature of MDE enabled by model formalization is the automatic model manipulation. This manipulation is called *model transformation*, in which a target model is the automatic generated from a source model, according to a transformation definition [86]. A model transformation takes one or more models as input, and generates one or more models as output according to mappings defined over the concepts of the input and output metamodels [139]. The input and output models do not necessarily conform to the same metamodel.

According to Czarnecki and Helsen [33], the main features of model transformations are: transformation rules, rule application scoping, source-target relationship, rule application strategy and rule scheduling. Next we explain each of them.

- **Transformation Rules.** A transformation rule is composed of a left-hand side (LHS) and a right-hand side (RHS). The left-hand side accesses the source model while the right-hand side considers the target model. The goal of these transformation rules is to describe a transformation process; in other words, it determines how a source model is transformed into a target model.
- **Rule Application Scoping.** Rule application scoping allows a transformation to restrict the parts of a model that participate in the transformation. The target scope of that transformation can be defined, which is the scope of the target model.
- **Source-Target Relationship.** The purpose of model transformations is to transform a source model into a target model. The target model can be a newly created model, based on information presented by the source model and the transformation rules, or it can be an updated version of the source model by updating, removing or adding elements to the source model.
- **Rule Application Strategy.** A rule application strategy is needed in order to determine where to apply a rule in a given source scope when more than one match is possible. The rule application strategy can be deterministic, non-deterministic or interactive.
- **Rule Scheduling.** This approach determines the order in which individual rules are applied. Rule scheduling can vary in the following main areas: form, rule selection, rule interaction and phasing.

At the top level, model transformations can be distinguished between model-to-text (M2T) and model-to-model (M2M) [33]. The distinction is that, while a model-to-model transformation creates its target as a model that conforms to the target metamodel, the target of a model-to-text transformation is essentially a string.

- **M2M transformations** provide the necessary support for translating models into other models, essentially by mapping source model elements into corresponding target model elements. In M2M transformation, both the source and target elements of the transformation are inside the MDE TS.

- **M2T transformations** are typically used to extract code from a model by defining code generations that produce text from models, such as documentation or textual representations of a model's content. It is the opposite of text-to-model (T2M) transformations that extract a model analyzing the source code.

M2T (called extractors) and T2M (called injectors) must deal with the different kinds of artifacts, that is, they must deal with the TS in which the artifact involved in the transformation are implemented. In the case of M2T transformations [32, 98], the vast majority of existing solutions allow these transformations to be defined through the use of templates, where the text to be generated conforms to a layout which is filled in by model information. For instance, we can extract and generate Java code from a UML model using a M2T transformation. On the other hand, solutions aimed at performing T2M transformations [32, 98] are normally ad-hoc tools that deal with a specific type of text artifacts. For instance, we need a parser to extract information from Java code and generate a UML model. The main difference between M2T and T2M is that template based M2T transformations can generate any text-based artifact (i.e., source code, XML, etc.) regardless of the formalism to which such an artifact conforms. However, the T2M transformations must know such formalism to be able to produce correct artifacts (correctness from the point of view of the conformance relationship).

In general, model transformations involve models in the sense of abstractions above program code. Since the concept of models is more general than the concept of program code, model transformations tend to operate on a more diverse set of artifacts than program transformations. In this scenario, model transformations can be classified according to the next categories [32, 98]:

- **Horizontal Model Transformation** is a transformation, where the source model and the target model belong to the same abstraction level. An example for this particular type of transformation is refactoring, because the target model marks a change in internal structure compared to the source model, but without altering the abstraction level of the model.
- **Vertical Model Transformation** is the opposite of horizontal model transformation. In this case, different levels of abstraction are used in the source and target model. It is also called refinement, because additional information is added, resulting in decreasing the abstraction level of the target model. On the other hand, information could be removed from the source model to create a more abstract target model.
- **Endogenous Model Transformation** is a transformation that affects models expressed in the same language. Both source and target models conform to the same metamodel. Examples for endogenous transformations include optimization or refactoring.
- **Exogenous Model Transformation** is the opposite of endogenous transformation. Exogenous transformations are expressed between models conforming to different metamodels. Exogenous transformations can also be called translations, because they focus on the mapping of a model in different metamodels. An important example for exogenous transformations is code generation, or its opposite, reverse engineering. During code generation, a model conforming to a specific metamodel is translated to its equivalent expressed in another metamodel.

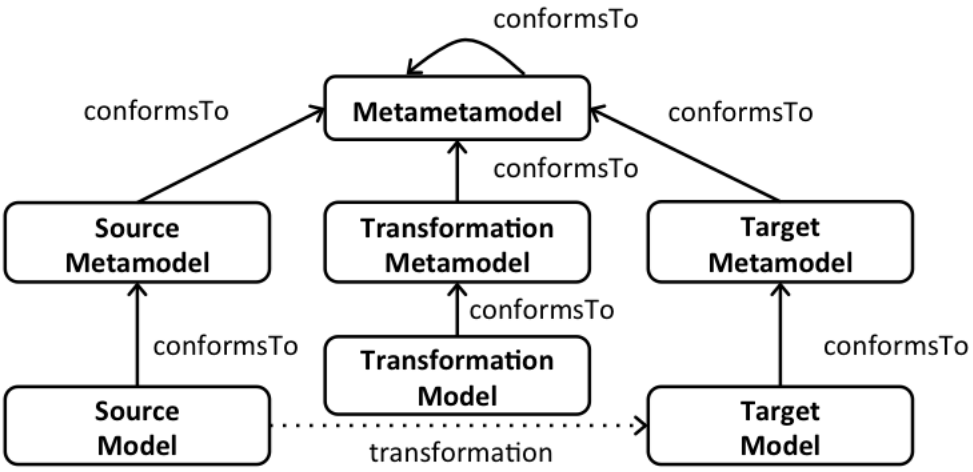


Figure 2.6: Model transformations approach using transformation models (adapted from Jouault et al. [76]).

A model transformation can be expressed in a model transformation language like ATL [76], QTV [56], Tefkat [94], Kermeta [38] or RubyTL [134], or in a general purpose language like Java, C, C#, etc. However, model transformation languages can offer advantages, such as syntax that makes it easy to refer to model elements. Model transformation languages are also based on a metamodel, and every written transformation can be regarded as an instance of a metamodel.

Figure 2.6 shows a model transformation that takes as input a model conforming to a given source metamodel and produces as output another model conforming to a target metamodel. In this case, a transformation can be expressed as a model that is called *transformation model*. Following this principle, it is possible to write a transformation that transforms a model into an instance of a transformation language metamodel, meaning that the output of the transformation is a transformation itself. This is called a higher-order transformation.

2.2.4 Higher-order transformations

Model transformations can either be written manually by a programmer or automatically generated using Higher-order Transformations (HOT). These are transformations modifying/reading/creating model transformations. A higher-order transformation is a model transformation such that its input and/or output models are themselves transformation models [156].

The generation of transformations is possible, since a transformation is also a model that conforms to some transformation metamodel [161, 155]. For instance a transformation written in ATL is indeed a model that conforms to the ATL metamodel. This uniformity has several benefits: it allows the reuse of tools and methods, and it creates a framework that can be applied recursively (since transformations of transformations can be transformed themselves). In this sense, Tisi et al. [155] identified four HOT patterns: transformation synthesis,

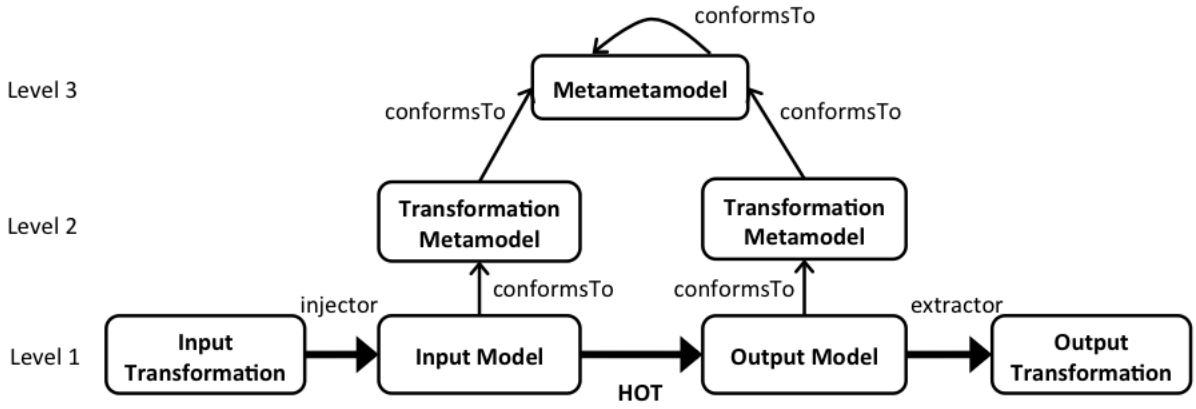


Figure 2.7: Higher-order transformation architecture for transformation modification [156].

transformation analysis, transformation composition and transformation modification. The transformation synthesis is the common pattern for HOTs that generate transformations from different information sources. The transformation analysis is the pattern for HOTs that take transformations as input, to generate different kinds of data, in the form of output models. The transformation composition is the pattern for HOTs that take multiple transformations as input and/or output. The transformation modification is the pattern for HOTs that take a transformation as input and generate a modified version of the same transformation.

HOT patterns are used for different applications; for instance, Figure 2.7 shows the typical schema of a HOT for transformation modification. These HOTs must have one transformation as input and one transformation as output. HOTs can also be classified according to the TS involved:

- *Injector.* The textual representation of the transformation rules is translated into a model representation. The generated model conforms to the transformation metamodel.
- *Higher-order transformation.* The transformation model is the input of a model transformation that produces another transformation model. In transformation modification the input, output and HOT transformation models all conform to the same transformation metamodel.
- *Extractor.* Finally an extraction is performed to serialize the output transformation model back into the output transformation model into textual transformation code.

2.2.5 Domain-specific languages and MDE

All engineering areas apply generic techniques together with others with more specific purposes. Applying generic techniques provides general solutions to several problems, even though these solutions may not be optimal. Specific techniques are usually optimized for the domain and provide a better solution for a reduced set of problems. This dichotomy also exists for programming languages and modeling: general-purpose and domain-specific languages.

A Domain-specific Language (DSL) is a language designed to be useful for a specific task (or set of tasks) in a certain problem-domain, as opposed to a general-purpose language [109]. Fowler, also called them micro or small languages; and he defines a DSL as a computer programming language of limited expressiveness focused on a particular domain [44]. These DSLs are programming or specification languages capable of being executed by a computer, and whose expressive power is focused and constrained to a problem domain [99]. Van Deursen et al. [159] provide a comprehensive definition, indicating that 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.

A DSL is defined by means of concrete and abstract syntax definitions, and possibly a semantics definition, which may be formulated at various degrees of preciseness and formality [44]. According to Voelter et al. [164], *abstract syntax* of a language is a data structure that holds the core information in a program, but without any of the notational details. The *concrete syntax* of a language is a textual or graphical representation that consists of a set of rules (productions) that define the way programs look to the programmer. Finally, the *semantics* captures the effect of sentences of the language and describes the meaning of each element.

A DSL may be classified as external or internal, depending on how it is designed and implemented. An external DSL is designed to be independent of any particular language. An internal DSL, on the other hand, is designed and implemented using a host language. Both kinds of DSLs have pros and cons: an external DSL gives the developer the freedom to design the syntax of a language in exactly the way he/she likes, and an internal DSL rides on the syntax of a host language, so the developer does not need to spend any time or effort worrying about compiling or parsing.

Although there exist DSLs for defining model transformations, these languages contain complex syntax and constructs that require advanced knowledge in these languages. A Domain-specific Transformation Language (DSTL) is a small, abstract, expressive and efficient language specifically designed to be applied in a certain domain and/or kind of transformations, if compared to other general transformations languages that have higher syntactic and semantic complexity [132]. Irazabal et al. [69] have proposed a DSTL implementation using MOFScript [108] (Model-to-text transformation language) for the databases domain. Izquierdo et al. [24] developed Gra2Mol using Java (general purpose language) for generating text-to-model transformations.

2.2.5.1 The notion of modelware

In MDE, metamodels can be used for defining new modeling languages for exchanging and storing information. This paradigm is the world where models live, and it is known as *modelware*. Modelware is a technical space centered on the concept of models. These models provide abstract representations of some concern of a software system, such as views, high-level designs, or implementation-level artifacts such as source code [171]. Each model conforms to a metamodel, which describes the language of the model. Metamodels are also

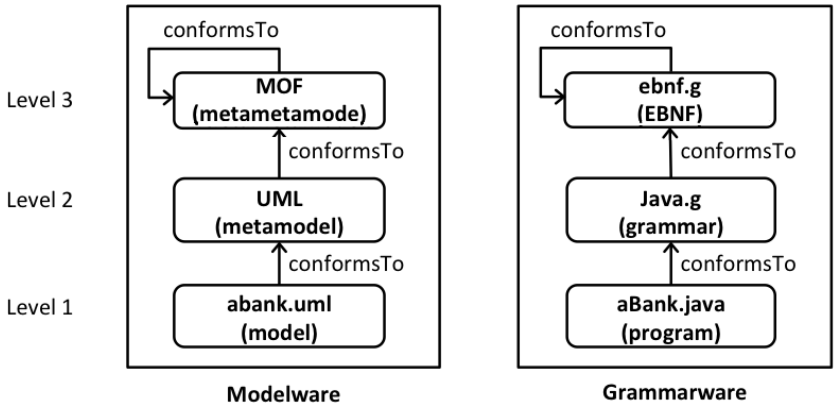


Figure 2.8: Modelware vs. Grammarware (adapted from Brambilla et al. [18]).

models and conform to their own metametamodel (e.g., ECore, KM3 or MOF). The benefit of describing everything in terms of models is that we can reason about the diversity of languages in a uniform manner.

Modelware is not different from grammarware (technical space where languages are defined by grammar) in terms of definition and architecture. Figure 2.8 shows the relationship between modelware and grammarware technical spaces. In grammarware, a language is defined in terms of grammars that conforms to Extended Backus–Naur Form (EBNF) metalanguage, which is a textual representation of the grammar rules of the language, and allows users to write programs. Analogously, in modelware a language can be defined in terms of metamodels that conform to a MOF metametamodel and allow users to define models that conform to metamodel. The model/metamodel relationship is equivalent to the relation between the grammar of a given programming language and a language to define grammars.

2.2.5.2 Domain-specific modeling

The Domain-specific Modeling (DSM) is a software engineering methodology for designing and developing systems, such as computer software. It involves systematic use of a domain-specific language to represent the various facets of a system [82]. DSM and DSML (domain-specific modeling language) are similar to the DSL concepts, but applied to the world of models. Whereas in DSLs the language (its textual representation) is domain specific, in DSM the models are domain specific.

DSM can raise the level of abstraction beyond coding by specifying programs directly using domain concepts. The final products can be generated from these high-level specifications. This automation is possible because the modeling language and generator only need to fit the requirements of one domain, often in only one company [81].

The DSML is a modeling language dedicated to a particular problem domain and/or a particular solution technique. DSMLs are closer to the problem domain and concepts than general purpose modeling languages such as UML. Moreover, DSMLs aimed at modeling and used for formalizing an application structure, behavior and requirements within a particular

domain. DSMLs have become very important within the field of MDE because each model is written in a specific language, and these models may be transformed into other models [85]. DSML can be developed using two approaches: create a DSML as an instance of a certain metamodel, or create a DSML by using some modeling language. The language used to create other modeling languages is called metalanguage.

A metamodel specifies a DSML. The metamodeling language consists of a set of elementary modeling concepts that represent the basic conceptual building blocks of any given approach and corresponding support tools. The metamodel defines fundamental concepts that may include composition, inheritance, various associations, attributes and other concepts. Moreover, the metamodel allows for specifying what concepts to include, how to combine them, and what editing operations should operate on them.

DSM enables developers to separate previously connected development activities for a software system. Thus, it allows them to concentrate on a single task at a time, which leads to better results [82]. In accordance to Czarnecki we identify the following three activities during development [32]:

- DSMLs are developed, reused or the existing ones are enhanced to express the desired models of the problem domain.
- Code generators are implemented. These components transform models to an executable solution.
- The project specific knowledge or problem description is expressed in the DSMLs, and the generators are used to map these models into a running solution.

Chapter 3

Supporting the Development of Tailoring Transformations

This chapter presents several tools developed in this thesis work, which help improve the MDE-based strategy for tailoring software processes proposed by Hurtado [62] that is part of the motivation of this thesis. These tools not only address some issues of such proposal, but also are part of the integrated solution for supporting the tailoring software processes approach presented in Chapter 4.

In order to introduce the Hurtado's proposal and the tools developed to deal with the unsolved issues, the next section presents a running example. Section 3.2 shows the MDE-based strategy for tailoring software processes using models and transformations proposed by Hurtado [62]. Section 3.3 presents a software context modeling tool, developed as part of this thesis work, which allows users to define the project context in two stages. Section 3.4 describes two projectors, which allow users to transform software processes into software process models, and vice versa. Section 3.5 describes the tailoring transformation and highlights potential challenges that are the motivation for the rest of this thesis.

3.1 Running Example

In order to illustrate the MDE-based tailoring strategy proposed by Hurtado [62], we will show how this approach is applied for tailoring software processes. For this purpose, we will illustrate the approach using part of the organizational software process of Rhiscom. Also, we use this organizational software process to show some of the tools developed for building the whole solution.

Rhiscom[1] is a software services company based in Santiago, Chile. It is a medium-sized company that develops integrated software and hardware solutions for the retail business. Employees perform more than one role in the company, according to traditional software

[1]Rhiscom website: http://www.rhiscom.com

Table 3.1: Decisions for Rhiscom's variable process elements.

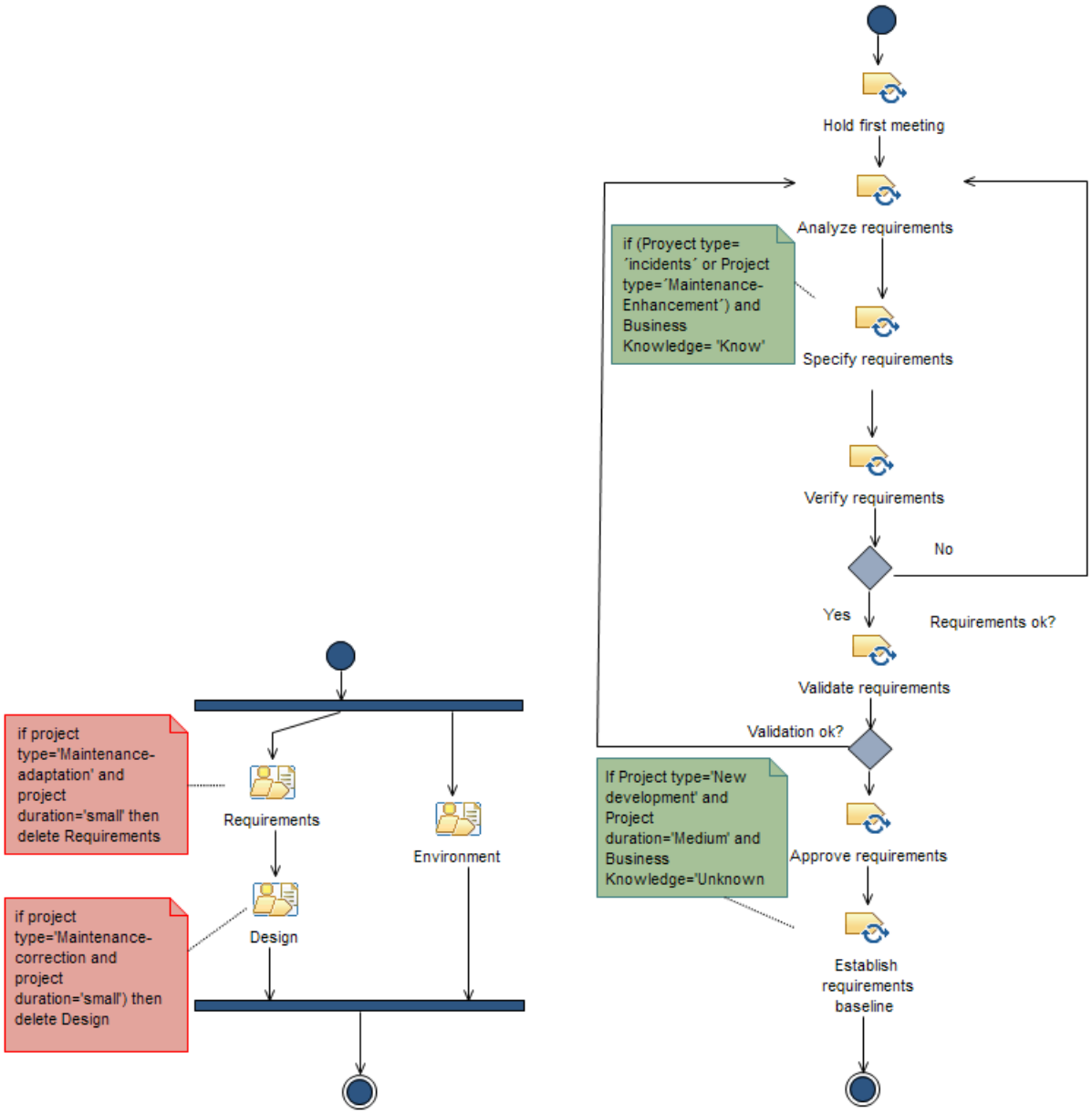
Decision	Process Element	Variability Type	Conditions	Conclusion
Decision 1	Requirements	Optional	(Project Type= Maintenance-adaptation) AND (Project Duration=Small)	Remove: Requirements
Decision 2	Design	Optional	(Project Type= Maintenance-correction) AND (Project Duration=Small)	Remove: Design
Decision 3	Specify Requirements	Alternative	(Project Type=Incidents OR Project Type= Maintenance-enhancement) AND (Business Knowledge=Know)	Replace: Specify Requirements in plain text
Decision 4	Establish Requirements Baseline	Alternative	Project Type= New development AND Project Duration=Medium AND Business Knowledge=Unknown	Replace: Establish Requirements Baseline and Test Cases

engineering disciplines (e.g., developer, analyst, tester, etc.). This company has a software process that was formalized using Eclipse Process Framework (EPF) [2] tool together with its variability, as part of the ADAPTE project[3].

The software development process is based on RUP and it is quite detailed in its definition. Figure 3.1a shows the *Elaboration* phase and Fig. 3.1b shows the *Requirements* activity. Note that variability is indicated in the activity diagram as a set of annotations, even though variability is actually specified using SPEM variability primitives in EPF. For instance, activities or tasks with red annotations denote optional elements that determine if they should be included or not in the adapted software process; and activities or tasks with green annotations denote alternative elements that establish if they should be replaced by another task or activity. These color annotations are not standard; they were defined by the company for their own use.

Table 3.1 describes four decisions for variable elements that are part of Rhiscom's software process. Each variable element has a set of conditions and a conclusion that decides on the variable element. For instance,the *Requirements* activity is optional and can be removed from the adapted software process for small maintenance projects (Decision 1). On the other hand, *Specify Requirements* is an alternative task and can be replaced to *Specify Requirements in plain text* in the adapted software process (Decision 3) for incidences or simple projects.

[2]EPF website: http://www.eclipse.org/epf/
[3]ADAPTE website: http://www.adapte.cl



(a) Rhiscom's *Elaboration* Phase

(b) Rhiscom's *Requirements* Activity

Figure 3.1: Rhiscom's Software Process in EPF (partial view).

3.2 The Hurtado Proposal

Although MDE techniques have been applied for tailoring software processes [90], particularly Hurtado et al. [65] proposed an MDE-based tailoring as a production strategy of project-specific process models in the context of a SPrL, which was the inspiration and the basis for this thesis. That tailoring proposal uses as input an organizational software process and the project context formally defined as models as shown in Figure 3.2. Consequently, the tailoring is implemented by means of a model transformation, whose inputs are the organizational software process model including variabilities, and the project context model.

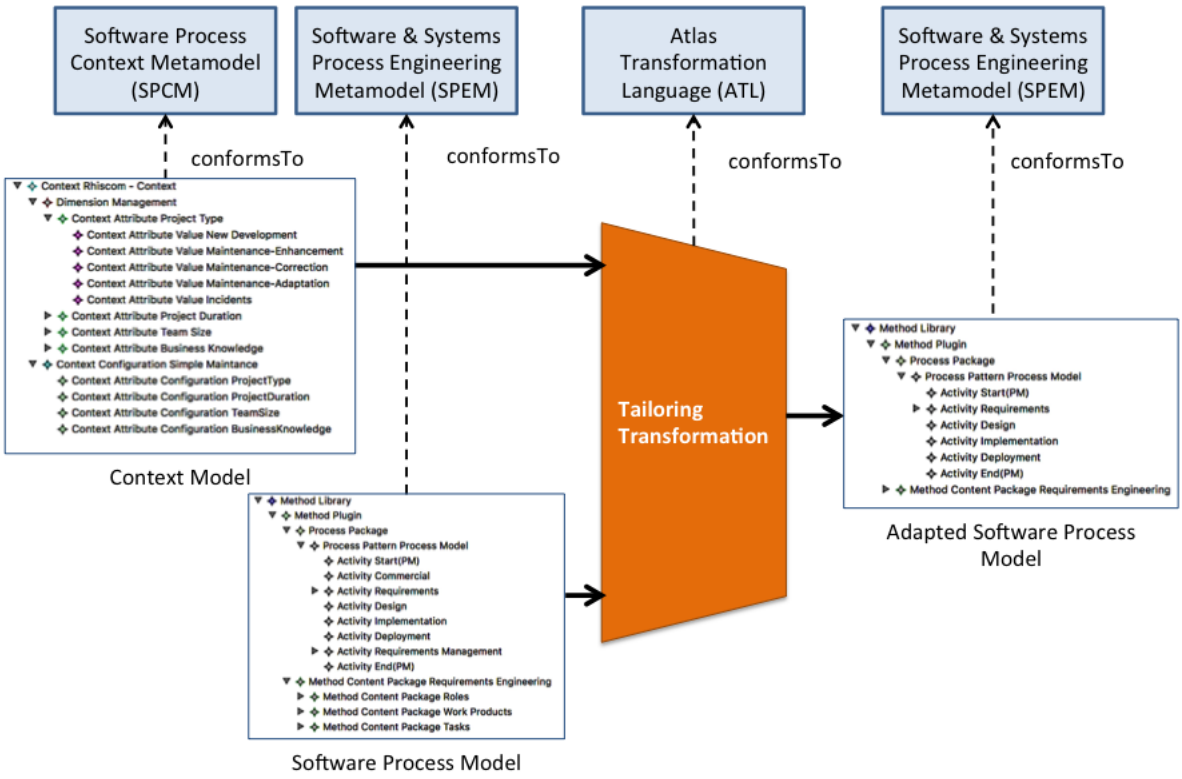


Figure 3.2: MDE-based Strategy for tailoring software processes (adapted from Hurtado et al. [65]).

Its output is the project-adapted software process model.

The organizational software process is defined as a *Software Process Model* conforming to the SPEM 2.0 [55] and the project context is defined as a *Context Model* conforming to Software Process Context Metamodel (SPCM) [65]. The *adapted software process model* is a SPEM 2.0 process model with variabilities resolved, i.e., with no variabilities. The transformation is implemented in ATL. The following subsections describe these components more in detail.

3.2.1 Organizational software process model

Hurtado et al. [65] defined experimental SPEM (eSPEM) that is a subset of SPEM 2.0 used for specifying the organizational software process model. eSPEM provides primitives for specifying software process models and also variability. Figure 3.3 shows part of the eSPEM metamodel.

A software process model that conforms to eSPEM is modeled as a method plug-in including *ProcessElements* and their linked *methodContentElements*. These *methodContentElements* specifically correspond to *TaskDefinitions* that have *WorkProductDefinitions* as input and output, and performed by (or participate with) *RoleDefinitions*. *ProcessElements* correspond to *RoleUse*, *TaskUse* and *WorkProductUse* that refer to activity-specific occurrences of

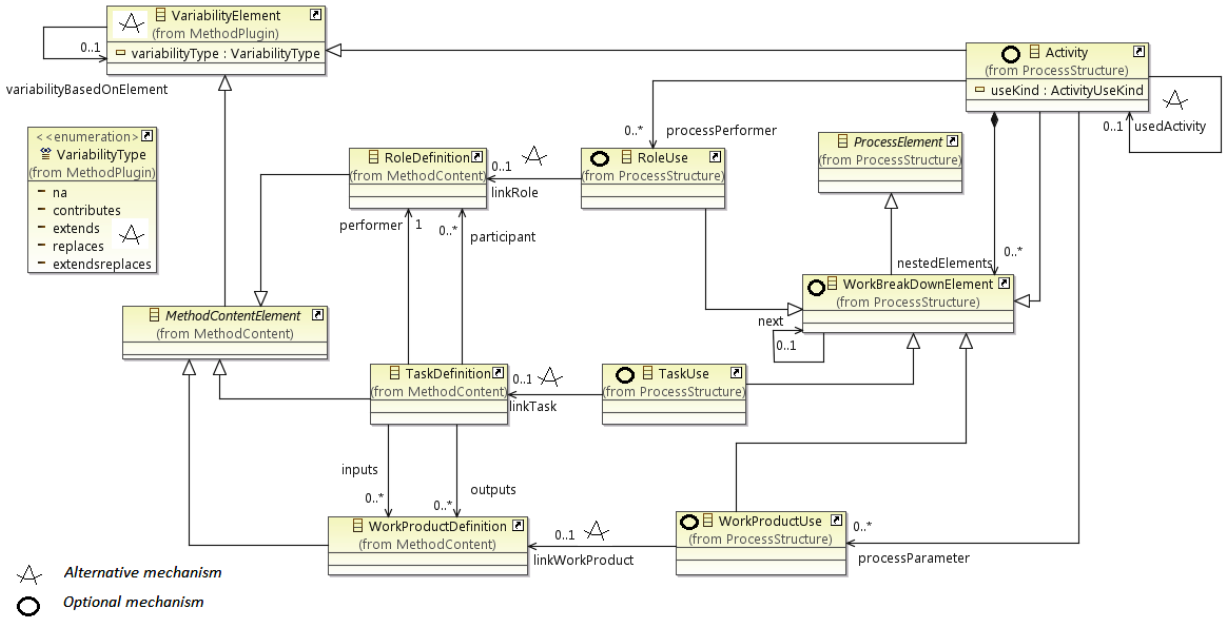


Figure 3.3: Part of experimental SPEM (eSPEM) highlighting where variability is specified [65]).

the respective *MethodContentElement*. The *ProcessElements* are organized in a *WorkBreakdownElements*.

The variability in the software process model can be either optional and/or alternative. A *ProcessElement* is optional when it can be removed from the software process and it is indicated using the isOptional attribute. A *ProcessElement* is alternative when it can be modified or extended by other *VariableElements* of the same kind according to a *VariabilityType* (extends, replaces, contributes, extends-replace). Therefore, each *MethodContentElement* (*TaskDefinition*, *RoleDefinition* and *WorkProductDefinition*) can be a *VariableElement*.

In Hurtado's proposal, software process models are defined conforming to eSPEM and they are edited using Exeed (Extended EMF Editor). Modeling and tailoring take place using the Eclipse Modeling Framework (EMF)[4] that allows defining models in a tree-like format and also executing transformations.

In order to use Hurtado's proposal, Rhiscom's software process needs to be defined as a formal model in eSPEM. In this sense, the process engineer needs to define an equivalent software process model in EMF environment to apply MDE-based tailoring. Figure 3.4 shows part of Rhiscom's software process model that was manually translated from the software process specified in EPF. The Method Plugin includes Process Package and Method Content Package. The Process Package has a Process Pattern that includes activities and tasks of Rhiscom's software process. *Requirements* activity is depicted in Fig. 3.4 and it has seven tasks: *Hold First Meeting*, *Analyze Requirements*, *Specify Requirements*, *Verify Requirements*, *Validate Requirements*, *Approve Requirements* and *Establish Requirements Baseline*. *Requirements* is highlighted and in the lower part of the figure we can see the properties available for

[4]EMF website: http://www.eclipse.org/modeling/emf/downloads/

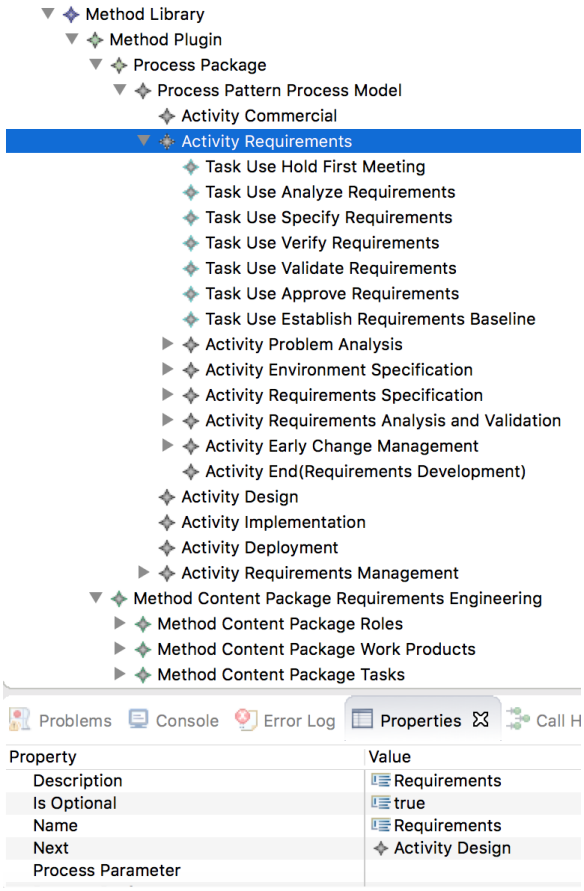


Figure 3.4: Rhiscom's eSPEM software process model.

defining complementary information; in this case *Requirements* is marked as optional. The Method Content Package has all definitions for tasks, roles and work products.

Even though eSPEM provides the formalisms required for MDE-tailoring, formally modeling software processes, it can be difficult to understand for process engineers and it is usually error prone. Moreover, most process engineers could hardly translate the software process specified in EPF into an eSPEM model.

3.2.2 Context model

The context of a project may vary according to the value of different project variables along different dimensions such as: size, duration, complexity, development team size and knowledge about the application domain. Hurtado et al. [65] formalize these characteristics as a context model that can be used to tailor the organizational software process model. This context model is defined by the Software Process Context Metamodel (SPCM) shown in Figure 3.5.

SPCM is based on three basic concepts: *ContextAttribute*, *Dimension* and *ContextConfiguration*. Every element in SPCM extends a *ContextElement*. A *ContextAttribute* represents

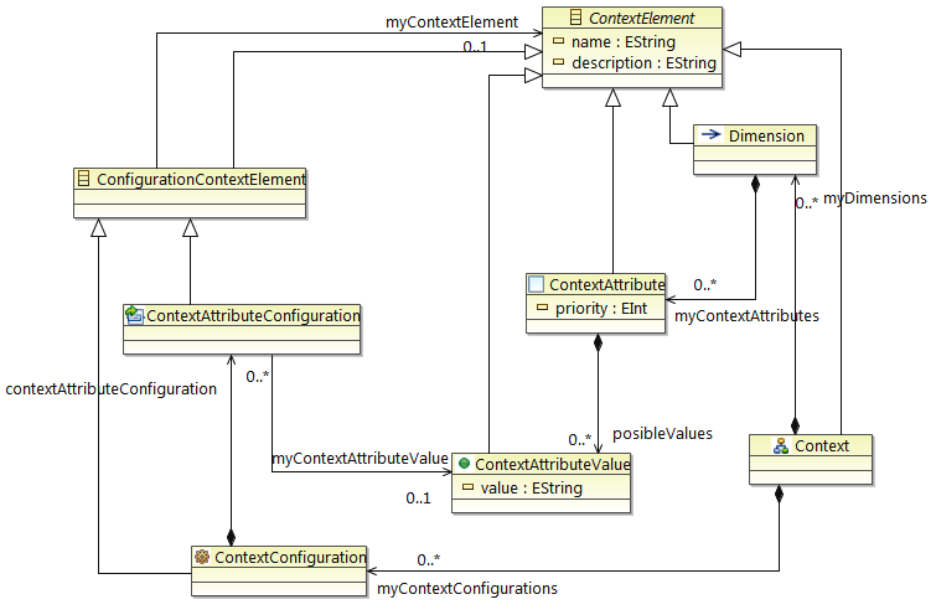


Figure 3.5: Software Process Context Metamodel (SPCM) [65]).

a relevant characteristic of the project context required for software process tailoring and can take one of a set of values defined as *ContextAttributeValue*. A *ContextAttributeValue* represents a type for qualifying a *ContextAttribute*. A *Dimension* represents a collection of related *ContextAttribute*. A *Context* is specified as a collection of *dimensions* and represents the whole context model. To represent possible specific process contexts, *ContextConfiguration* can be defined from the context model. A *ContextConfiguration* is a collection of *ContextAttributeConfiguration* that is set to one of the possible *ContextAttributeValue* for each *ContextAttribute*. Therefore, a *ContextAttributeConfiguration* is associated with *ContextAttribute* and also with one unique *ContextAttributeValue*.

In Hurtado's proposal, context models are defined conforming to SPCM, and they are edited using EMF. In order to define Rhiscom's project context, the process engineer needs to define both, a context model and the project manager needs to configure a project context in the EMF environment. Figure 3.6 shows Rhiscom's context model that was manually built, which includes two sections: *Context* that includes all context attributes and their potential values and *Context Configuration* that configures the context attributes with a particular value for defining a project context. The *Context* section has one *Management* dimension that includes four context attributes: *Project Type*, *Project Duration*, *Team Size* and *Businesses Knowledge*. The first context attribute may be either *New Development*, *Maintenance-Enhancement*, *Maintenance-Correction*, *Maintenance-Adaptation* or *Incidents*, and the last one may be *Known*, *Affordable* or *Unknown*. The *Context Configuration* section includes a list of all context attributes that can be configured with a particular value for defining a project context. The context attribute *Project Type* is highlighted and in the lower part we can configure the *ProjectType* as *Maintenance-Correction* as well as the other context attributes.

However, formally modeling project contexts, which is usually required to perform this tailoring activity, also represents a challenge for potential users; and particularly for small

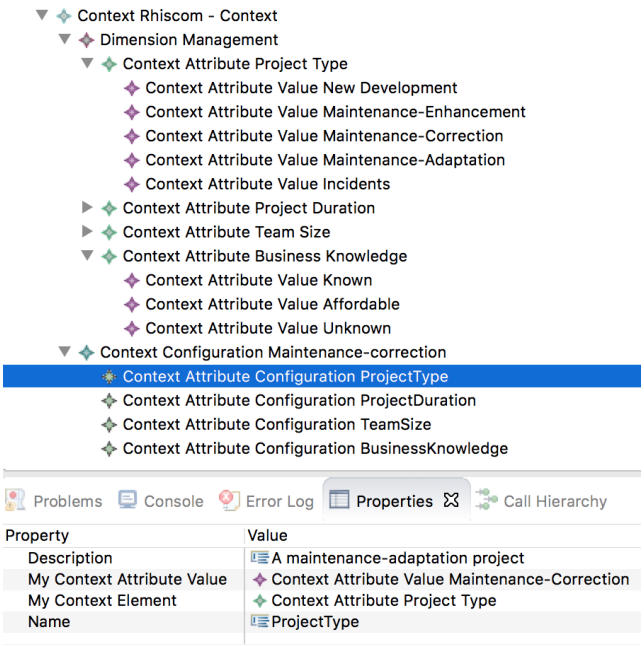


Figure 3.6: Rhiscom's context model.

and medium-sized software companies, which usually have low expertise and resources to deal with formal methods and notations.

In order to address this difficulty, we developed a user friendly tool that allows defining the context in two sequential activities [110]. The first activity is performed by the process engineer, and its goal is to define the relevant dimensions, attributes and potential values of an *organizational context model*. The second activity is performed by the project manager, and its goal is to configure the previously defined organizational context model as a particular *project context model*. These two activities are coordinated so that only those attributes defined as part of the organizational context model can be configured for a project. The output of this tool is the *organizational context model* represented through an XML file and the *project context model* represented through an XMI file that is used as one of the inputs for the process tailoring transformation. In this way, users do not have to deal with EMF or any other modeling environment in order to count on the context model required for MDE-tailoring. Next we present the main features of this tool.

3.3 Software Context Modeling Tool

The software context modeling tool allows process engineers to perform two key activities through a graphical user interface: the organizational context model definition and project context model configuration. Next we explain how these activities are conducted and also present the main user interfaces that allows performing them.

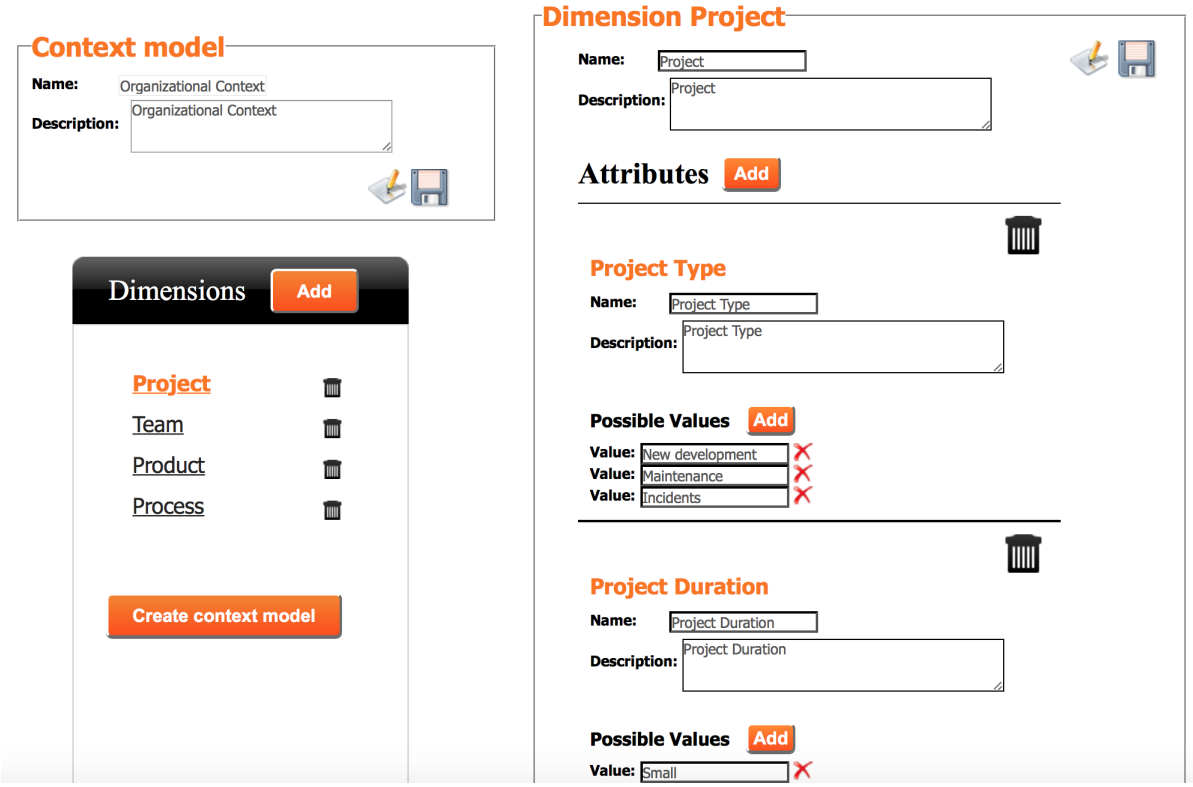


Figure 3.7: Defining the organizational context model.

3.3.1 Organizational context model definition

The organizational context model is the specification of the context attributes and their sets of potential context attributes values that allow characterizing a particular project context. For example, a *Project Type* that is a context attribute can have three potential values: *New Development*, *Maintenance* and *Incidents*, and a *Project Duration* can have three potential values: *Small*, *Medium*, *Large*. The context attributes can be grouped in different dimensions for better comprehension, e.g., a *Project* dimension can include two context attributes: Project Type and Project Duration.

The organizational context model that conforms to SPCM can be defined manually using EMF in XML Metadata Interchange (XMI) format[5]. This activity is not highly complex, but it is not user friendly for software process engineers. In order to maximize usability, we developed a graphical interface for defining the organizational context model that conforms to SPCM and can be visualized in EMF.

Figure 3.7 shows part of the user interface that allows the process engineer to define the organizational context model. The tool provides a canonical specification that has been obtained from empirical experiences in software process definition and improvement. The canonical context proposes four dimensions and their attributes: *Project*, *Team*, *Product* and *Process*. Based on that, the tool presents this canonical organizational context model as a starting point, and the process engineer can then adjust it according to the needs of the

[5]XMI website: http://www.omg.org/spec/XMI/

organization. This Web tool is available at the ADAPTE project website[6].

The tool has a context section in the top-left of the user interface that allows process engineers to edit (service available by clicking edit icon) and save (save icon) the name and description of the context model being defined. Moreover, the tool has a dimension section at the bottom-left of the user interface that allows process engineers to create (add button), remove (trash icon) and update (using the name and description text field in the top-right) context dimensions.

The tool also has an attributes section (shown in the right part of the figure) that allows process engineers to create (add button), remove (trash icon) and update (name and description text field) context attributes belonging to a context dimension. Moreover, process engineers can create, remove and update context attributes values for each context attribute. Finally, the organizational context is specified as a model, and it is generated as an XML file by clicking on the "create context model" button.

Every software company can interpret each context value in a different way, because different companies can use the same attribute or value name with different meanings. These companies have to determine which context variables and dimensions are relevant for them (i.e., for their projects). For instance, a certain company may get engaged in different types of project, such as *New Development* or *Maintenance*. In that case, the *Incidents Project Type*, as established by the canonical context model, does not make sense for them and it should be removed. The canonical context model does not only act as a guideline for process engineers, but also contributes to homogenizing the jargon used for project dimensions, variables and values within each company.

Figure 3.8 shows an example of an organizational context model defined using the organizational context modeling tool. The canonical context model has been modified according to Rhiscom's organizational context. In this case, Rhiscom's organizational context only has the *Management* dimension that includes four context attributes (*Project Type*, *Project Duration*, *Team Size* and *Business Knowledge*) and their potential values. For instance, *Project Type* can assume five possible values: *New Development*, *Maintenance-Enhancement*, *Maintenance-Correction*, *Maintenance-Adaptation* and *Incidents*.

3.3.2 Project context model configuration

Each particular software project that the company develops has its own characteristics, and these characteristics determine the concrete software process that best fits it; therefore, they should be identified. The project characteristics are specified by instantiating the context variables of the organizational context model. For example, a concrete project context may consider six attributes and respective their values: *Project Type* is *New Development*, *Project Duration* is *Medium*, *Team Size* is *Medium*, *Technical Complexity* is *High*, *Quality Type* is *High* and *Process Focus* is *Final Product*.

As mentioned before, the project context model definition is not a highly complex activity,

[6]ADAPTE website: http://adapte.dcc.uchile.cl:8080/Context/contextModelHome.html

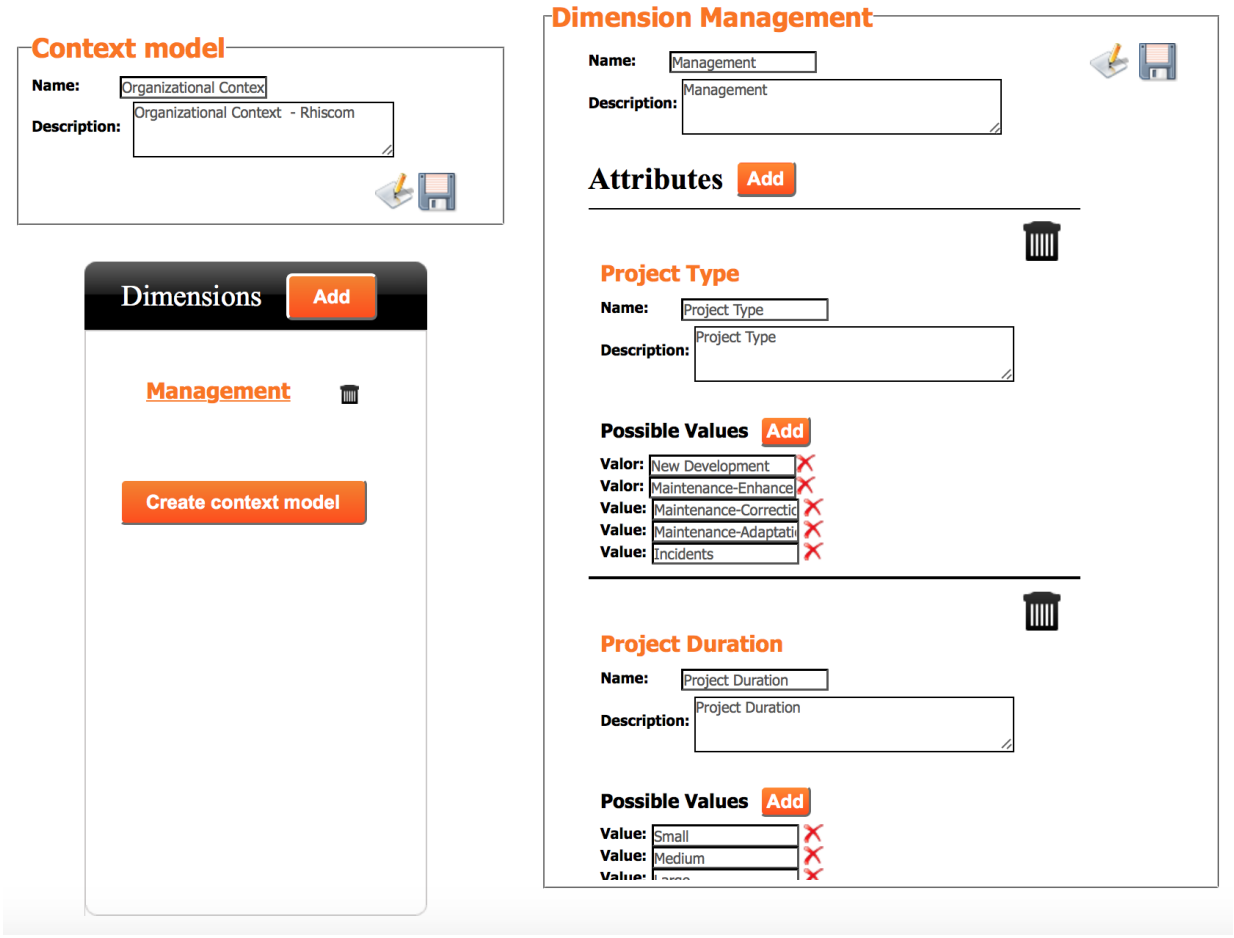
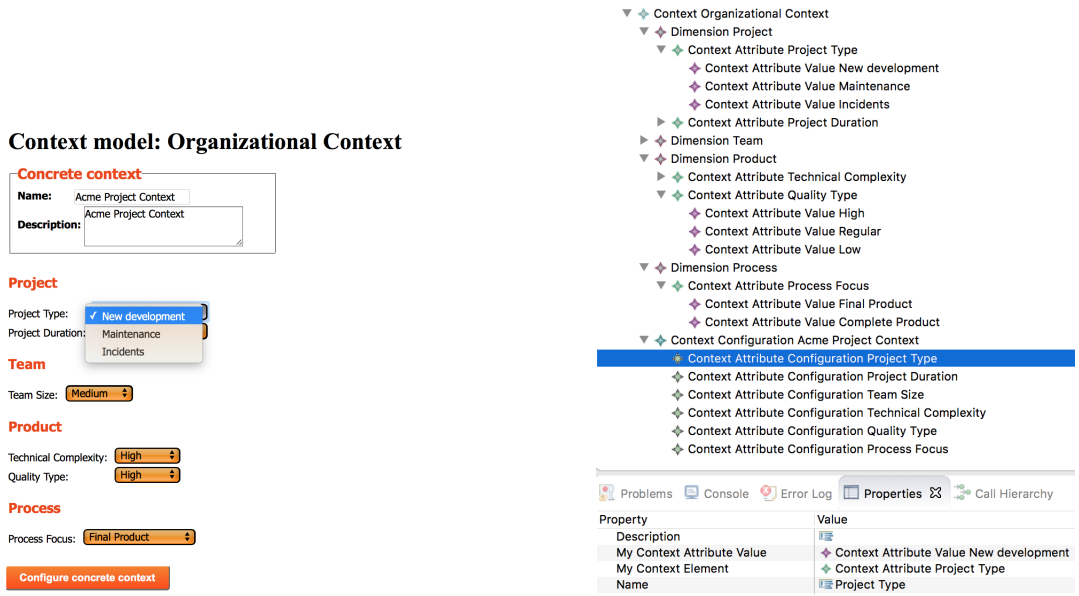


Figure 3.8: Rhiscom's organizational context model.

but do it is not simple for project managers if they have to do it manually. Therefore, we extended the previous tool to support the specification of particular project contexts (i.e., their features) through a graphical user interface, and thus to ease such an activity. This specification activity requires configuring the project context model, using as input the organizational context model shown in the previous section. This configuration tool allows project managers to set the attributes established by the organizational context model, and also select the appropriate value in each case (i.e., according to the features of the project to be addressed).

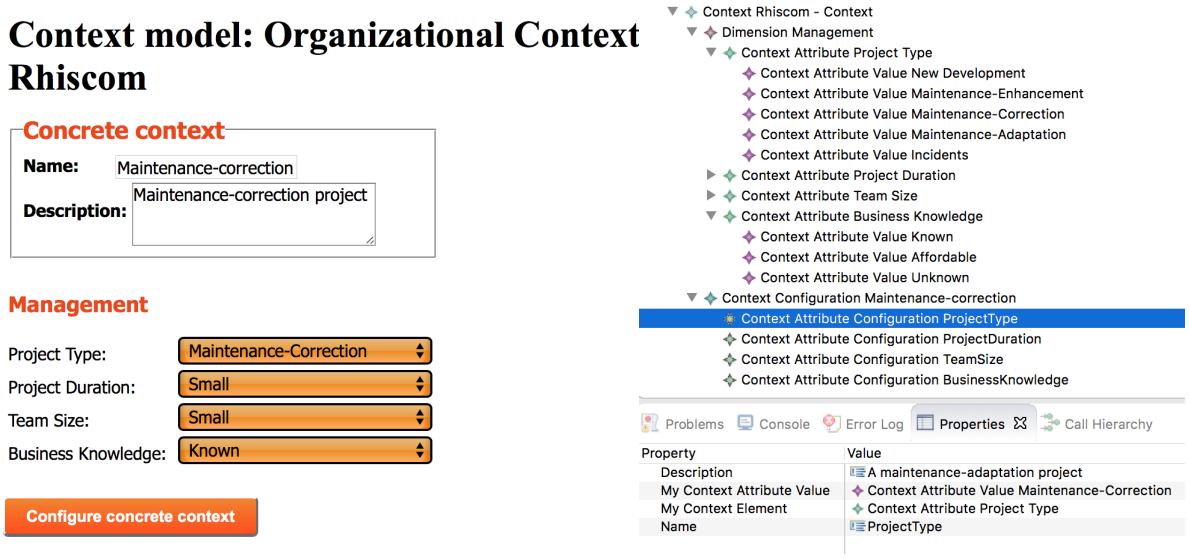
The tool has a concrete context section for defining the name and description of the context model. Also, the tool has a configuration section at the bottom-left of the user interface that allows project managers to define a particular project context by setting the attributes values of the organizational context for the project at hand. Figure 3.9a shows part of the user interface for defining the project context model.

Figure 3.10a shows a concrete (project) context model of Rhiscom, based on the organizational context model defined in Fig. 3.8. In this case, the Rhiscom's concrete project context has the following characteristics: *Project Type* is *Maintenance-Correction*, *Project Duration* is *Small*, *Team Size* is *Small* and *Business Knowledge* is *Known*. Figure 3.10b shows the resulting project context model (called context model in MDE-based tailoring) that is required



(a) Project context (b) Project context model

Figure 3.9: Defining the project context model.



(a) Project context (b) Project context model

Figure 3.10: Rhiscom's project context model.

as input for the tailoring transformation together with the software process model.

Although the formal representation of the context model is specified in XMI format, end-users (i.e., the process engineer and the project manager) only interact with typical web user interfaces to create, adjust and configure this model. This eases considerably such an activity and makes it more affordable, particularly for small and medium-sized software companies that have non-specialized human resources. This tool was designed to support both, the process engineer during the definition of the project attributes that may affect the process to be applied, and also the project manager when assigning the particular values to these

attributes depending on the project at hand. This tool has been successfully verified using already defined and validated formal context models in several Chilean software companies.

3.4 Transformations between Software Process and Software Process Model

In the MDE-based tailoring approach proposed in [65], the organizational software process needs to be defined as a model so that transformations can be applied. Modeling and tailoring take place using the Eclipse Modeling Framework (EMF)[7], which is an environment that allows defining models and transformations, and also executing transformations. The organizational software process model that conforms to eSPEM can be defined manually using EMF in XML Metadata Interchange (XMI)[8] format. Performing manually this activity is not very difficult, but it is error prone. Moreover, once the organizational software process model has been tailored, its model format results difficult to understand for users (process engineers) in the tree-like format provided by EMF. Therefore, the users could hardly validate the results of the tailored process, let alone make use of the benefits of the tailored process without translating it back to its graphical format.

Eclipse Process Framework implements most part of the SPEM standard from a user point of view, and it can import/export the specified software processes as XML files, so that processes can be externally manipulated. Even though EPF implements the SPEM 2.0, the data is represented internally conforming to a different metamodel, i.e., Unified Method Architecture (UMA) metamodel.

Projectors are used to implement either model-to-text or text-to-model transformations, and they receive the name of extractor and injectors, respectively [75]. The difference between a projector and a traditional transformation is that while transformations define a correspondence between elements from the source and target metamodels, the projectors do not count on a metamodel in either side of the equation, source or target, but elements of the grammar of a certain language, so matching should be defined in a different fashion. One of the widest uses of extractors is generating source code from a model.

[7]EMF website: http://eclipse.org/modeling/emf/
[8]XMI website: http://www.omg.org/spec/XMI/

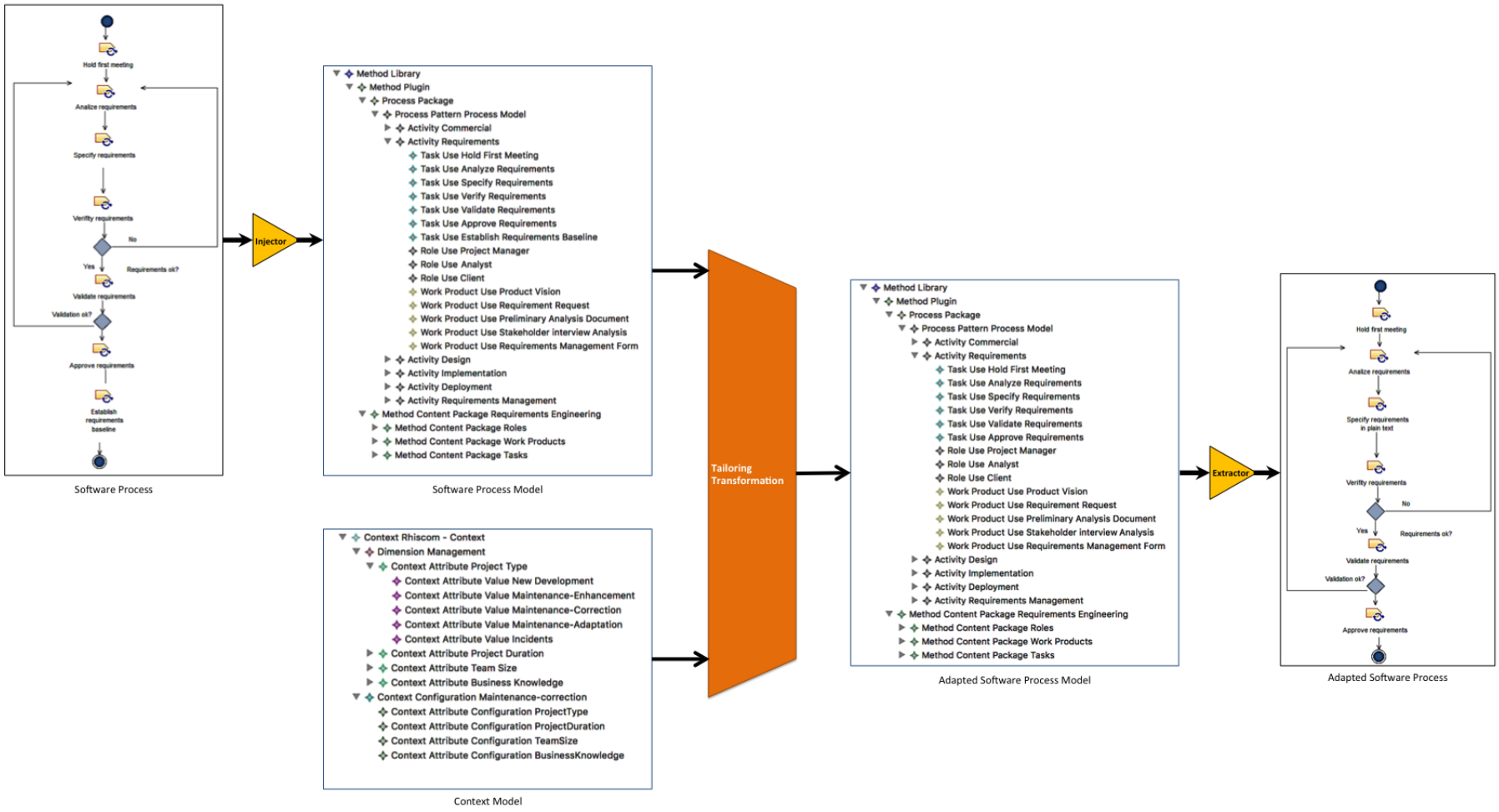


Figure 3.11: Text-to-model, model-to-model, and model-to-text transformations involved in software processes tailoring.

As part of this thesis work we developed two projectors that allow potential users to apply transformations between software process and software process model [10]. These projectors transform a software process from EPF (specified as an XML file) into a software process model that conforms to eSPEM in XMI format and vice versa. The translation back and forth is now transparent, and the potential users can easily understand and validate the tailored process as depicted in Fig. 3.11. Therefore, projectors play an important role in the whole tailoring process in order to improve its usability, and the understandability of the results.

MDE technology allows for formally manipulating concepts as models with a high level of abstraction. However, some of these models may require alternate representations in another technical space [11]; therefore, it may be necessary to represent an artifact both as a model and as a non-model. On the one hand, a model can be manipulated via model transformations. On the other hand, a non-model can be generated via generative programming. It can also be more appropriately manipulated either manually by a developer or automatically by existing external tools. Bridging those two worlds, projectors allow generating a model from external representation and vice versa.

There are several Domain-specific Transformation Languages (DSTL) designed for model-to-text and text-to-model transformations. Textual Concrete Syntax (TCS) [75] is a particular technology for performing such kind of transformations[9]. Here the structure of the textual elements is specified by its grammar, while the metamodel is used for the model, and a mapping is specified between these two worlds. There are other languages for implementing projectors, such as EGL (Epsilon Generation Language) [130], JET[10] and Xpand [84]. Although these languages are generally used for code generation, they are not the best choice for two directional projectors since they are not easy to maintain by our potential users (i.e., process engineers and project managers). Similar to other transformations, projectors can also be implemented in general purpose languages like Java, C++ or C#.

3.4.1 Projectors

3.4.2 Projectors implementation

As mentioned before, in this thesis work we developed two projectors: an injector and an extractor. The first one transforms the software process that is exported by EPF (XML file that conforms to the UMA metamodel), into a software process model (XMI file that conforms the eSPEM metamodel). The second projector (i.e., the extractor) transforms the adapted software process model back into an adapted software process, allowing thus to visualize it in EPF and validate it with the users (see Fig. 3.11).

[9]TCS website: http://www.eclipse.org/gmt/tcs/
[10]JET website: http://www.eclipse.org/modeling/m2t/?project=jet

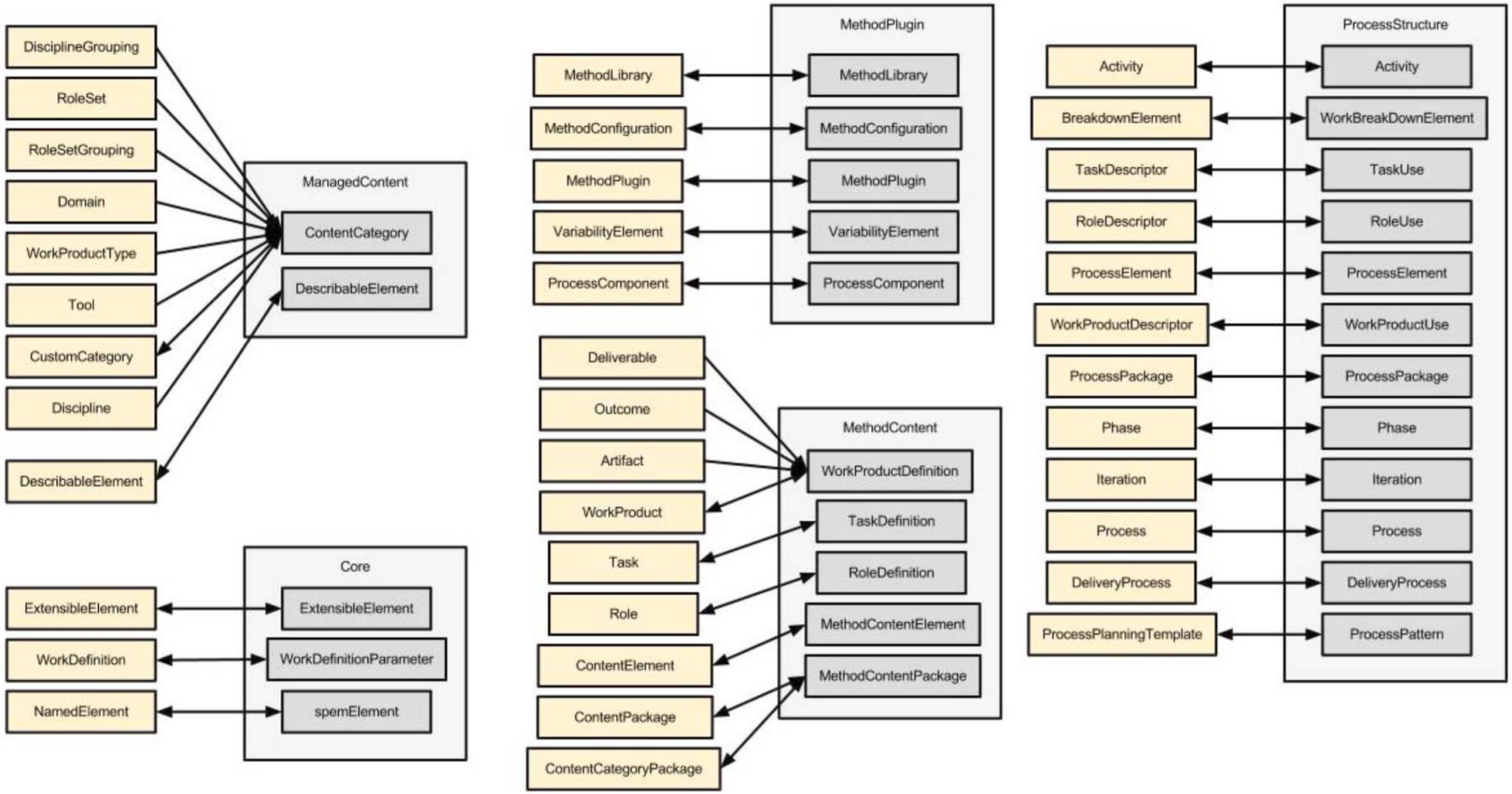


Figure 3.12: Relationship between UMA (yellow boxes) and eSPEM elements (gray boxes).

Table 3.2: Matching between UMA and eSPEM elements of ManagedContent.

UMA Element	**eSPEM Element**	**Observation**
DisciplineGrouping	ContentCategory	ContentCategory represents the generalization of UMA Elements.
RoleSet		
RoleSetGrouping		
Domain		
WorkProductType		
Tool		
Discipline		
CustomCategory	ContentCategory	CustomCategory and ContentCategory are equivalent.
DescribableElement	DescribableElement	

UMA and eSPEM have some similar process elements, but there are other process elements that are not present in both metamodels. Figure 3.12 shows the relationship between UMA (yellow boxes) and eSPEM elements (gray boxes). The UMA metamodel is much larger than the eSPEM metamodel (123 vs. 31 classes), and thus it is expected that there is information loss when transforming the XML file into XMI. This is not a problem in itself, but some of the classes originally in the UMA process that are not part of the obtained eSPEM process, may be needed afterwards when the adapted software process is to be visualized back in EPF, i.e., the extractor should be able to restate these lost process elements. Consequently, the extractor needs the original software process (XML file) to complete the lost process elements.

Table 3.2 shows the matching between UMA and eSPEM elements belonging to the ManagedContent package. The ManagedContent meta-model package introduced concepts for managing the textual content of such descriptions. These concepts can either be used standalone or in combination with process structure concepts. Considering the UMA and eSPEM specification, we can define that *ContentCategory* of eSPEM represents the generalization of UMA elements: *DisciplineGrouping*, *RoleSet*, *RoleSetGrouping*, *Domain*, *WorkProductType*, *Tool*, *Discipline* and *Custom Category*. Moreover, the *CustomCategory* and *ContentCategory* are equivalent.

Table 3.3 shows the matching between UMA and eSPEM elements belonging to the Core package. The Core meta-model package contains the meta-model classes and abstractions that build the fundamental elements for all other meta-model packages. Considering the UMA and eSPEM specification, we can define that UMA elements *ExtensibleElement*, *WorkDefinition* and *NamedElement* are semantically equivalent to the following eSPEM elements: *ExtensibleElements*, *WorkDefinitionParameter*, *spemElement*.

Table 3.4 shows the matching between UMA and eSPEM elements of MethodPlugin package. The MethodPlugin meta-model package provides the concepts for SPEM 2.0 users and organization to build up a development knowledge base that is independent of any specific process and development project. Considering the UMA and eSPEM specification, we can define that the concepts of UMA and eSPEM are the same for building the software process structure: *MethodLibrary*, *MethodConfiguration*, *MethodPlugin*, *VariabilityElement* and *ProcessComponent* (notice that the name of elements are the same).

Table 3.3: Matching between UMA and eSPEM elements of Core.

UMA Element	**eSPEM Element**	**Observation**
ExtensibleElement	ExtensibleElement	
WorkDefinition	WorkDefinitionParameter	WorkDefinition and WorkDefinitionParameter are equivalent.
NamedElement	spemElement	NamedElement and spemElement are equivalent.

Table 3.4: Matching between UMA and eSPEM elements of MethodPlugin.

UMA Element	**eSPEM Element**	**Observation**
MethodLibrary	MethodLibrary	
MethodConfiguration	MethodConfiguration	
MethodPlugin	MethodPlugin	
VariabilityElement	VariabilityElement	
ProcessComponent	ProcessComponent	

Table 3.5: Matching between UMA and eSPEM elements of MethodContent.

UMA Element	**eSPEM Element**	**Observation**
Deliverable	WorkProductDefinition	WorkProductDefinition represents the generalization of UMA Elements.
Outcome		
Artifact		
WorkProduct	WorkProductDefinition	WorkProduct and WorkProductDefinition are equivalent.
Task	TaskDefinition	Task and TaskDefinition are equivalent.
Role	RoleDefinition	Role and RoleDefinition are equivalent.
ContentElement	MethodContentElement	ContentElement and MethodContentElement are equivalent.
ContentPackage	MethodContentPackage	MethodContentPackage represents the generalization of UMA Elements.
ContentCategoryPackage		

Table 3.5 shows the matching between UMA and eSPEM elements of MethodContent package. The MethodContent meta-model package provides concepts for defining lifecycle and process-independent reusable method content elements, which provide a base of documented knowledge of software development methods, techniques, and best practices. MethodContent comprises of textual step-by-step explanations, describing how specific fine-granular development goals are achieved, by which roles, and with which resources and results, regardless of the placement of these steps within a specific development lifecycle. Considering the UMA and eSPEM specification, we can define that *WorkProductDefinition* and *MethodContentPackage* of eSPEM represent the generalization of the following UMA elements: *Deliverable*, *Outcome*, *Artifact*, *ContentPackage*, and *ContentCategoryPackage*, respectively. Moreover, the UMA elements *WorkProduct*, *Task*, *Role* and *ContenElement* are semantically equivalent to the following eSPEM elements: *WorkProductDefinition*, *TaskDefinition*, *RoleDefinition*, and *MethodContentElement*.

Finally, Table 3.6 shows the matching between UMA and eSPEM elements belonging to

Table 3.6: Matching between UMA and eSPEM elements of ProcessStructure.

UMA Element	**eSPEM Element**	**Observation**
Activity	Activity	
BreakdownElement	WorkBreakDownElement	BreakdownElement and WorkBreakDownElement are equivalent.
TaskDescriptor	TaskUse	TaskDescriptor and TaskUse are equivalent.
RoleDescriptor	RoleUse	RoleDescriptor and RoleUse are equivalent.
ProcessElement	ProcessElement	
WorkProductDescriptor	WorkProductUse	WorkProductDescriptor and WorkProductUse are equivalent.
ProcessElement	ProcessElement	
ProcessPackage	ProcessPackage	
Phase	Phase	
Iteration	Iteration	
Process	Process	
DeliveryProcess	DeliveryProcess	
ProcessPlanningTemplate	ProcessPattern	ProcessPlanningTemplate and ProcessPattern are equivalent.

the ProcessStructure package. The ProcessStructure meta-model package provides concepts for defining a structure of the software process. Considering the UMA and eSPEM specification, we can define that the UMA elements *Activity*, *BreakdownElement*, *TaskDescriptor*, *RoleDescriptor*, *ProcessElement*, *WorkProductDescriptor*, *ProcessElement*, *ProcessPackage*, *Phase*, *Iteration*, *Process*, *DeliveryProcess* and *ProcessPlanningTemplate* are semantically equivalent to the eSPEM elements as shown in Table 3.6.

The projectors implemented in this thesis are Java applications that conduct both a text-to-model (T2M) and a model-to-text transformation (M2T). For this purpose, UMA and eSPEM metamodel elements are transformed into Java objects. The Java objects have methods that allow setting/getting information in the object (attributes and child nodes from XML/XMI file) by using an XML/XMI node (expressed in an *org.w3c.dom.Element* object).

The Java classes *org.w3c.dom.Document* and *org.w3c.dom.Element* were used for storing temporary elements during translation. Figure 3.13 shows the pipeline architecture followed by the projectors. The blue arrows —from the XML file to the XMI file-– represent the flow for the injector, while the red arrows —from the XMI file to the XML file-– illustrate the extractor's flow.

Document (org.w3c.dom.Document) represents an XML or XMI file. *Document* is used to get an *Element* object using *Element element=document.getDocumentElement()*, or set an *Element* (see line 8 in Listing 3.1). The *Document* implements methods for getting and setting nodes, attributes and values using JDOM[11]. JDOM is a Java library for accessing, manipulating, and outputting XML data. Notice that XMI is a specific application of XML and can be manipulated using JDOM. Listing 3.1 shows two methods for parsing a UMA (XML) file to eSPEM (XMI) file representation and vice versa: *parseDocsUmaToSpem* (see

[11]JDOM website: http://www.jdom.org

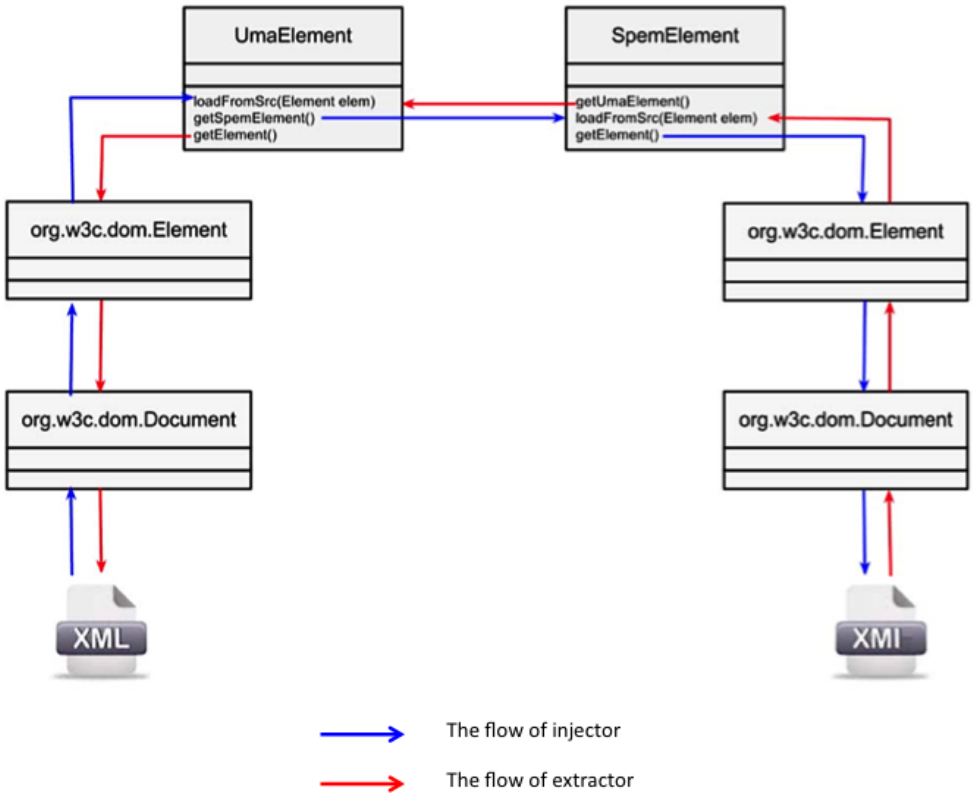


Figure 3.13: Pipeline architecture of the projectors.

line 1 in Listing 3.1) and *parseDocsSpemToUma* (see line 17 in Listing 3.1).

Listing 3.1: Excerpt of the *Document* implementation in Java.

```
1  public void parseDocsUmaToSpem(String umaFileToParse, String spemTargetFile, String encoding) {
2          Document umaDoc = DocumentUtils.getDocumentFromSrc(umaFileToParse);
3          Document spemDoc = new DocumentImpl();
4          if (umaDoc == null || spemDoc == null) {
5          return;
6          }
7          ReferenceObject reference = new ReferenceObject(spemDoc, umaDoc);
8          Element src = umaDoc.getDocumentElement();
9          UmaMethodLibrary toParse = new UmaMethodLibrary(reference);
10         toParse.loadFromSrc(src);
11         SpemMethodLibrary parsed = parseUmaToSpem(toParse);
12         Element trg = parsed.getElement();
13         String toPrint = ElementUtils.printElement(trg, encoding);
14         StringUtils.printToFile(encoding, spemTargetFile, toPrint);
15 }
16
17 public void parseDocsSpemToUma(String spemFileToParse, String umaOriginalFile, String umaTargetFile,
       String encoding) {
18         Document spemDoc = DocumentUtils.getDocumentFromSrc(spemFileToParse);
19         Document umaDoc = new DocumentImpl();
20         Document umaOriginalDoc = DocumentUtils.getDocumentFromSrc(umaOriginalFile);
21         if (spemDoc == null || umaDoc == null || umaOriginalDoc == null) {
22         return;
23         }
24         ReferenceObject reference = new ReferenceObject(spemDoc, umaDoc);
25         ReferenceObject referenceOriginal = new ReferenceObject(spemDoc, umaOriginalDoc);
26         Element src = spemDoc.getDocumentElement();
27         Element srcOriginal = umaOriginalDoc.getDocumentElement();
28         SpemMethodLibrary toParse = new SpemMethodLibrary(reference);
29         toParse.loadFromSrc(src);
30         UmaMethodLibrary original = new UmaMethodLibrary(referenceOriginal);
31         original.loadFromSrc(srcOriginal);
32         UmaMethodLibrary parsed = parseSpemToUma(toParse, original);
33         Element trg = parsed.getElement();
34         String toPrint = ElementUtils.printElement(trg, encoding);
35         StringUtils.printToFile(encoding, umaTargetFile, toPrint);
36 }
```

Element (org.w3c.dom.Element) represents a node in the XML or XMI file. *Element*

extends the *Node* class (*org.w3c.dom.Node*) and contains a name (*getNodeName()*), a value (*getNodeValue()*), and a content (*getTextContent()*). Also, *Element* can get attributes of a *Node* object in a *NamedNodeMap* (*org.w3c.dom.NamedNodeMap*) using *NamedNodeMap attributes = element.getAttributes()* method. Moreover, the *Element* gets and sets all node children elements using *NodeList listOfChils = element.getChildNodes()*. Listing 3.2 shows two methods for getting eSPEM and UMA elements: *getSpemElement* (see line 1 in Listing 3.2) and *getUmaElement* (see line 20 in Listing 3.2).

Listing 3.2: Excerpt of the *Element* implementation in Java.

```
1  public SpemMethodLibrary getSpemElement(String nodeName) {
2          SpemMethodLibrary result = new SpemMethodLibrary(reference);
3          result.setId(id);
4          result.setNodeName(nodeName);
5          result.setName(name);
6          result.setDescription(briefDescription);
7          List<SpemMethodPlugin> ownedMethodPlugins = new ArrayList<SpemMethodPlugin>();
8          for (UmaMethodPlugin mp : methodPlugins) {
9                  ownedMethodPlugins.add(mp.getSpemElement("ownedMethodPlugin"));
10         }
11         result.setOwnedMethodPlugin(ownedMethodPlugins);
12         List<SpemMethodConfiguration> predefinedConfigurations = new ArrayList<SpemMethodConfiguration>();
13         for (UmaMethodConfiguration pc : methodConfigurations) {
14                 predefinedConfigurations.add(pc.getSpemElement("predefinedConfiguration"));
15         }
16         result.setPredefinedConfigurations(predefinedConfigurations); return result;
17 }
18
19 public UmaMethodLibrary getUmaElement(String nodeName) {
20         UmaMethodLibrary result = new UmaMethodLibrary(reference);
21         result.setId(id);
22         result.setNodeName(nodeName);
23         result.setName(name);
24         result.setBriefDescription(description);
25         List<UmaMethodPlugin> methodPlugins = new ArrayList<UmaMethodPlugin>();
26         for (SpemMethodPlugin omp : ownedMethodPlugin) {
27                 methodPlugins.add(omp.getUmaElement("MethodPlugin"));
28         }
29         result.setMethodPlugins(methodPlugins);
30         List<UmaMethodConfiguration> methodConfigurations = new ArrayList<UmaMethodConfiguration>();
31         for (SpemMethodConfiguration pc : predefinedConfigurations) {
32                 methodConfigurations.add(pc.getUmaElement("MethodConfiguration"));
33         }
34         result.setMethodConfigurations(methodConfigurations); return result;
35 }
```

Transformer (javax.xml.transform.Transformer) generates an XML or XMI from a *Document*. The transformer needs to initialize using *TransformerFactory (javax.xml.transform.TransformerFactory)*. The transformer implements a mapping between UMA and eSPEM elements using *Map<String, UmaElement>* and *Map<String, SpemElement>*. Moreover, we developed a template that is used by the transformer for building a eSPEM model (XMI) or a UMA file (XML). The template considers encoding, indentation and recursive methods for building and validating a model or file (see Listing 3.3).

Listing 3.3: Excerpt of the *Transformer* implementation in Java.

```
1  public static Element baseUmaElement(Document doc, String nodeName) {
2          Element element = doc.createElement(nodeName);
3          element.setAttribute("xmlns:xsi", "http://www.w3.org/2001/XMLSchema-instance");
4          element.setAttribute("xmlns:uma", "http://www.eclipse.org/epf/uma/1.0.6");
5          return element;
6          }
7
8  public static Element baseSpemElement(Document doc, String nodeName) {
9          Element element = doc.createElement(nodeName);
10         element.setAttribute("xmi:version", "2.0");
11         element.setAttribute("xmlns:xmi", "http://www.omg.org/XMI");
12         element.setAttribute("xmlns:xsi", "http://www.w3.org/2001/XMLSchema-instance");
13         element.setAttribute("xmlns:methodContent", "http://spem/1.0/MethodContent");
14         element.setAttribute("xmlns:methodPlugin", "http://spem/1.0/MethodPlugin");
15         element.setAttribute("xmlns:processStructure", "http://spem/1.0/ProcessStructure");
16         element.setAttribute("xsi:schemaLocation", "http://spem/1.0/MethodPlugin "
17         + "http://spem/1.0/MethodContent#//MethodPlugin "
18         + "http://spem/1.0/ProcessStructure "
19         + "http://spem/1.0/MethodContent#//ProcessStructure");
20         return element;
21 }
```

The transformations (T2M and M2T) are executed in Java using the following syntax:

<mode> <input-file> <output-file> <original-file>

where *mode* is the execution mode for parsing XMI and XML files.

There are two modes ("uma2spem" and "spem2uma") that transforms UMA to eSPEM (T2M) and eSPEM to UMA (M2T), respectively. The variable *input-file* indicates the path to the input file, and *output-file* has the path to the output file. Fanally, *original-file* is only used in "spem2uma" mode, and it allows for the recovery of UMA elements, which were not considered in "uma2spem" mode.

For example, we can use the injector for generating a software process model that conforms to eSPEM, from an XML file that conforms to UMA using the following instruction:

uma2spem OriginalUMAProcess.xml eSPEMProcess.xmi

On the other hand, we can use the extractor for generating a software process that conforms to UMA, from an XMI file that conforms to eSPEM using the following instruction:

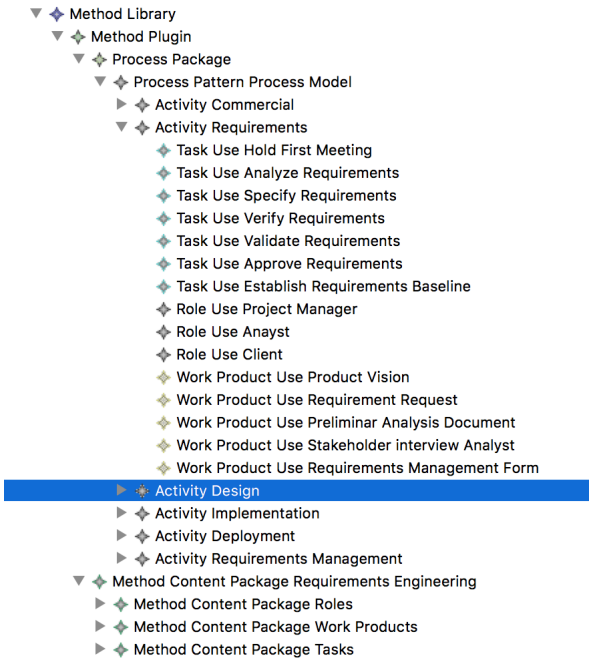
spem2uma eSPEMProcess.xmi OtherUMAProcess.xml OriginalUMAProcess.xml

Rhiscom's software process can be exported as an XML file from EPF (see Fig. 3.1a and 3.1b in Sec. 3.1). The injector (uma2spem) takes this XML file as input and generates the software process model that conforms to eSPEM. Figure 3.14a shows the software process model that is automatically generated and that can be visualized using EMT. On the other hand, Fig. 3.14b shows the adapted software process model that does not include the *Design* activity (notice that this activity is highlighted in Fig. 3.14a). Moreover, we can see that both software process models also include more information about other software process elements, as *RoleUse* and *WorkProductUse*.

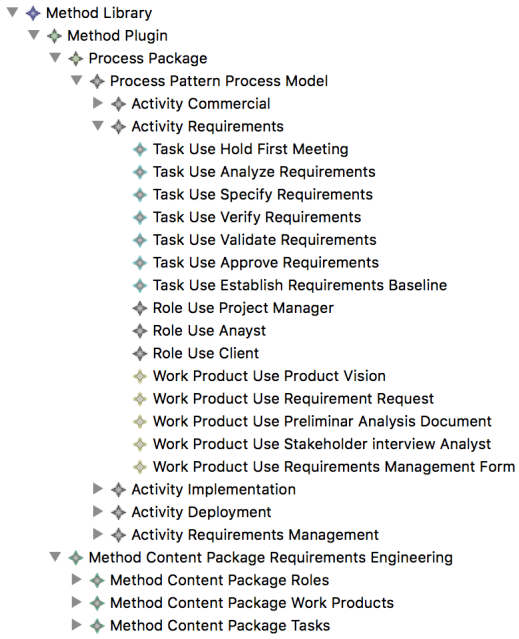
The extractor (spem2uma) takes as input the XMI file conforming to eSPEM that results from the tailoring transformation, and generates the adapted software process. Figures 3.15a and 3.15b show the software process that is automatically generated and it can be visualized from the EPF. We can see that the *Design* task has also been removed from the software process shown in EPF.

3.5 Tailoring Transformation

In MDE-based tailoring, the organizational software process model is specified in SPEM 2.0, and includes its potential variability. The variability of the software process may apply to: activities, tasks, roles and work products, and can be optional or alternative. Their variability can be established by setting to *true* the value of the Boolean attribute *isOptional*, and alternatives are specified by linking variation points to possible realizations using the *replaces* SPEM 2.0 variability primitive. On the other hand, as the organizational software process can be specified in UMA, we just use UMA constructs for specifying SPEM variability



(a) Rhiscom's software process model that includes the Design activity



(b) Rhiscom's adapted software process model without the Design activity

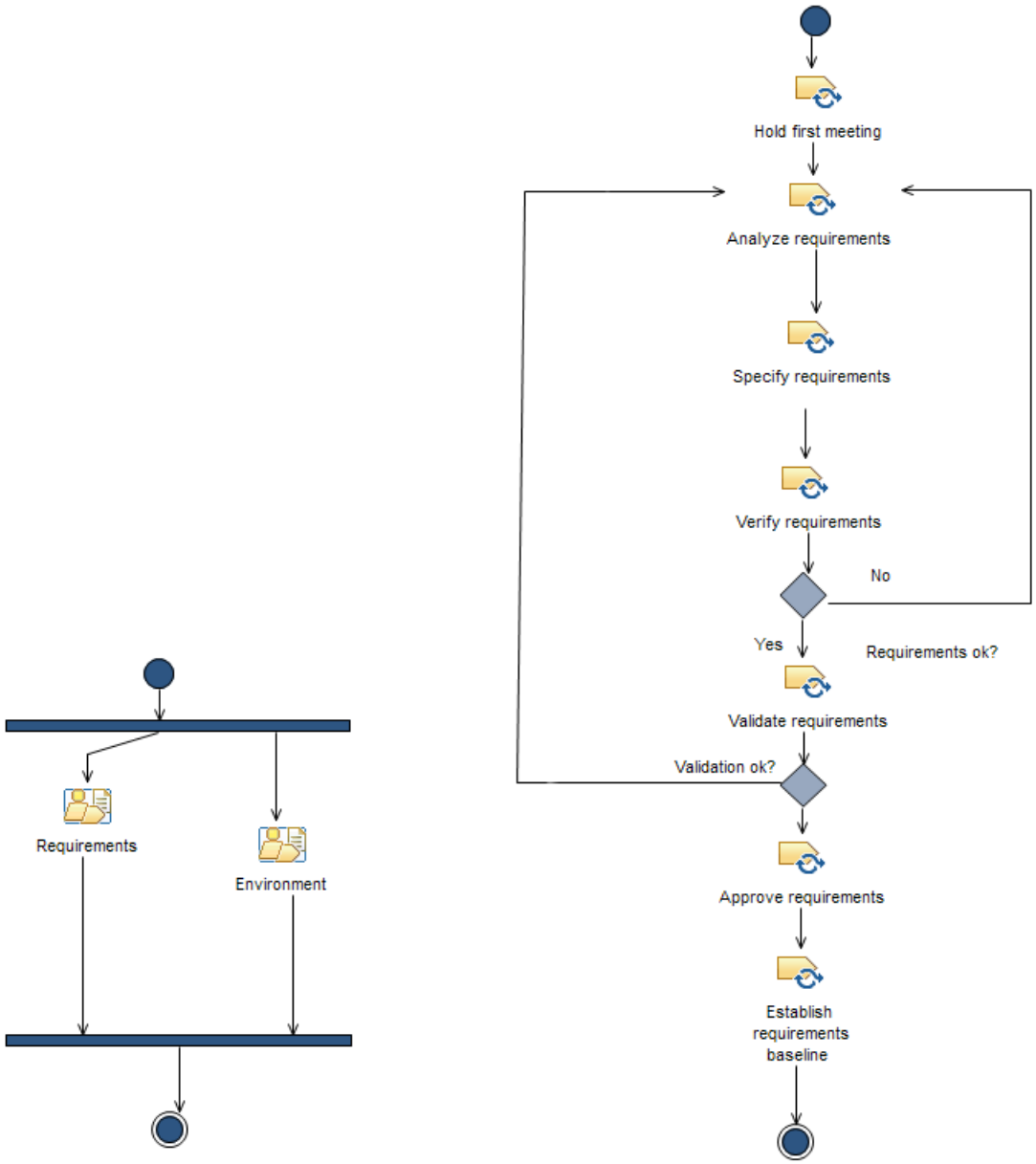
Figure 3.14: Rhiscom's software process model in EMT.

primitives.

Tailoring rules are the set of transformation rules about how to tailor the organizational software process. Tailoring rules decide on a variable process element using conditions and conclusions. Hurtado et al. [65] define tailoring rules within a tailoring transformation. The tailoring transformation is programmed in ATL [76] and allows for tailoring the organizational process model according to the values of different context configurations in the context model. The proposed tailoring transformation is endogenous [32] because its output conforms to the same eSPEM metamodel as the input.

The tailoring transformation in ATL is represented as a module that is divided into a *header* and a *body* section. The header states the name of the transformation module and declares the source and target models, which are typed by their metamodels. The body is composed of a set of rules as *matched rules* and helpers as *called rules*, which are stated in arbitrary order after the header section. Matched rules describe how the model should be generated from the source model. Called rules are auxiliary functions that enable the possibility of factorizing the ATL code used in different points of the transformation.

The matched rules allow us to specify: (i) which target elements should be generated for each source element, and (ii) how generated elements are initialized from the matched source elements. The helpers implement tailoring rules as ATL rules with their conditions, and they decide on variable process elements. Each variable process element has an associated helper called from the matched rule. Figure 3.16 shows rule *TaskUse*. The source pattern *MM!TaskUse* is defined after the keyword *from*, meaning that the rule will generate target elements for each source element matching the pattern. To select only those optional source



(a) Rhiscom's *Elaboration* phase

(b) Rhiscom's *Requirements* activity

Figure 3.15: Rhiscom's adapted software process in EPF.

elements that are relevant for a specific project, an extra condition is added: an *OptionalTailoringRule* implemented as a helper function. When this rule returns false, the element is removed from the process.

In eSPEM's variability mechanisms, a process element (e.g., *TaskUse*) could be linked to several alternative variants of method elements (e.g., *Task Definition*). An *AlternativeTailoringRule* is defined as a rule that returns the selected method element according to the helper rule. The *AlternativeTailoringRule* chooses the most suitable *TaskDefinition* alternative variant, according to the attribute value in the context (e.g., *Domain Knowledge*). If there were more variability points, a conjunction of rules would be applied, also specifying priorities to make trade-offs.

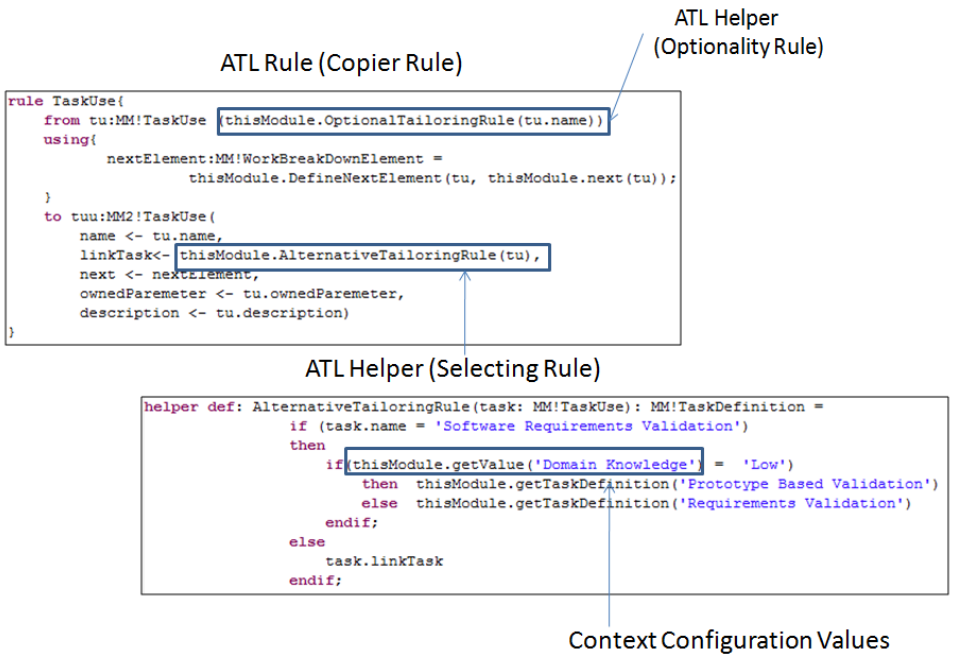


Figure 3.16: An example of the tailoring transformation in ATL [65].

Listing 3.4 shows an excerpt of the tailoring transformation that includes the header and body sections, and it also shows two helper rules that are part of Rhiscom's tailoring transformation: *optionalRule* (line 25). The transformation implements two optional rules involving *Requirements* and *Design* activities, and *alternativeRule* (line 45) that implements two alternative rules involving the *Establish Requirements Baseline* and *Specify Requirements* tasks. For example, *optionalRule* states that if "Project Type" is "Maintenance-Correction" and "Project Duration" is "Small", then the corresponding process element (in this case, the *Design* activity) will be omitted (*false*) from the adapted software process, otherwise it will be included (*true*). On the other hand, *alternativeRule* states that if "Project Type" is "New-Development", "Project Duration" is "Medium", and "Business Knowledge" is "Unknown", then the *Establish Requirements Baseline and Test Cases* task should replace the *Establish Requirements Baseline* task in the development process.

Listing 3.4: Rhiscom's tailoring transformation [65].

```
1  -- HEADER
2  -- Module:
3  -- Input and output metamodels
4  module Tailoring;
5  create OUT : MM2 from IN : MM, IN1 : MM1;
6
7  -- BODY
8  -- Helpers: Called Rules
9  -- To obtain the context attributes
10 helper def: getContextAttributeConfiguration (nameAttribute: String) :
11 MM1!ContextAttributeConfiguration = MM1!ContextAttributeConfiguration.allInstances()->asSequence()->select(
       a | a.myContextElement.name =nameAttribute )->first();
12 helper def: getValue(nameAttribute: String):
13 String = thisModule.getContextAttributeConfiguration(nameAttribute).myContextAttributeValue.value;
14
15 helper def:getTaskDefinition(taskDefinitionName:String): MM!TaskDefinition =
16 MM!TaskDefinition.allInstances()->asSequence()->select(t|t.name = taskDefinitionName)->first();
17
18 -- Helpers: Called Rules
19 -- To obtain the next WorkBreakDownElement
20 helper def:nextElement(a:MM!WorkBreakDownElement): MM!WorkBreakDownElement = MM!WorkBreakDownElement.
       allInstances()->select(t|t=a.next)->first();
21 helper def:next(a:MM!WorkBreakDownElement): MM!WorkBreakDownElement = if(thisModule.optionalRule(thisModule
       .nextElement(a).name)) then a else thisModule.next(thisModule.nextElement(a)) endif;
22
23 -- Helpers: Called Rules
24 -- To describe transformation rules
```

```
25 helper def: optionalRule(name:String): Boolean =
26         if(Sequence{'Requirements','Design'}.includes(name)) then(
27                 if('Requirements'= name) then
28                         {
29                         if(thisModule.getValue('Project Type') = 'Maintenance-Adaptation' and thisModule.
                                getValue('Project Duration') = 'Small') then  false
30                         else  true
31                         endif;
32                         }
33                         else (
34                                 if('Design'= name) then
35                                 {
36                                 if(thisModule.getValue('Project Type') = 'Maintenance-Correction' and
                                        thisModule.getValue('Project Duration') = 'Small') then  false
37                                 else  true
38                                 endif;
39                                 }
40                         )
41                 )
42         else  true
43         endif;
44
45 helper def: alternativeRule(tu:MM!TaskUse): MM!TaskDefinition =
46         if(Sequence{'Establish Requirements Baseline','Specify Requirements'}.includes(tu.name)) then(
47                 if('Specify Requirements'= tu.name) then
48                         {
49                         if((thisModule.getValue('Project Type') = 'Incidents' or thisModule.getValue('
                                Project Type') = 'Maintenance-Enhancement') and thisModule.getValue('Business
                                Knowledge') = 'Known') then thisModule.getTaskDefinition('Specify Requirements
                                 in plain text')
50                         else thisModule.getTaskDefinition(tu.name)
51                         endif;
52                         }
53                 else (
54                         if('Establish Requirements Baseline'= tu.name) then
55                                 {
56                                         if(thisModule.getValue('Project Type') = 'New Development' and
                                                thisModule.getValue('Project Duration') = 'Medium' and
                                                thisModule.getValue('Business Knowledge') = 'Unknown') then
                                                thisModule.getTaskDefinition('Establish Requirements Baseline
                                                and Test Cases')
57                                         else thisModule.getTaskDefinition(tu.name)
58                                         endif;
59                                 }
60                         else tu.linkTask
61                         endif
62                 ) endif
63         ) else tu.linkTask
64         endif;
65
66 -- Rules: Matched Rules
67 -- to map model elements
68 rule main{
69         from     ml:MM!MethodLibrary
70         to               mll:MM2!MethodLibrary(
71         name <- ml.name,
72         description <- ml.description,
73         ownedMethodPlugin<- ml.ownedMethodPlugin,
74         predefinedConfiguration<-ml.predefinedConfiguration
75         )
76 }
77 rule methodplugin{
78         from     mp:MM!MethodPlugin
79         to               mpp:MM2!MethodPlugin (
80         name<- mp.name,
81         description <- mp.description,
82         ownedProcessPackage<- mp.ownedProcessPackage,
83         ownedMethodContentPackage <- mp.ownedMethodContentPackage
84         )
85 }
86 rule methodconfiguration{
87         from c:MM!MethodConfiguration
88         to       cc:MM2!MethodConfiguration(
89         name <-c .name,
90         description <- c.description,
91         baseConfiguration<-c.baseConfiguration,
92         methodPluginSelection<-c.methodPluginSelection,
93         defaultView<-c.defaultView,
94         processView<-c.processView,
95         processPackageSelection<-c.processPackageSelection,
96         myProcessPackage<-c.myProcessPackage,
97         contentSelection<-c.contentSelection
98         )
99 }
100 rule ProcessPackage{
101         from pp:MM!ProcessPackage
102         to ppp:MM2!ProcessPackage
103         (
104         name <- pp.name,
105         processElements <- pp.processElements,
106         processPackages <- pp.processPackages
107         )
108 }
109 rule Activity{
110         from a:MM!Activity(
111         thisModule.optionalRule(a.name)
```

```
112         )
113         to aa:MM2!Activity(
114         name <- a.name,
115         nestedElements <- a.nestedElements,
116         processPerformer <- a.processPerformer,
117         processParameter <- a.processParameter,
118         usedActivity <- a.usedActivity,
119         useKind <- a.useKind,
120         next <- a.next,
121         ownedParemeter <- a.ownedParemeter,
122         description <- a.description,
123         variabilityType <- a.variabilityType,
124         variabilityBasedOnElement <- a.variabilityBasedOnElement)
125 }
126 rule TaskUse{
127         from tu:MM!TaskUse(thisModule.optionalRule(tu.name))
128
129         using{
130                 task:MM!TaskDefinition = thisModule.alternativeRule(tu);
131         }
132
133         to tuu:MM2!TaskUse(
134         name <- tu.name,
135         linkTask<- task,
136         next <- tu.next,
137         ownedParemeter <- tu.ownedParemeter,
138         description <- tu.description)
139 }
140 rule RoleUse{
141         from ru:MM!RoleUse
142         to ruu:MM2!RoleUse(
143         name <- ru.name,
144         linkRole<-ru.linkRole,
145         myPerformer <- ru.myPerformer,
146         ownedParemeter <- ru.ownedParemeter,
147         description <- ru.description
148         )
149 }
150 rule WorkProductUse{
151         from wpu:MM!WorkProductUse
152         to wpuu:MM2!WorkProductUse(
153         name <- wpu.name,
154         linkWorkProduct<-wpu.linkWorkProduct,
155         linkedProcessElement <- wpu.linkedProcessElement,
156         ownedParemeter <- wpu.ownedParemeter,
157         description <- wpu.description
158         )
159 }
160 rule MethodContentPackage{
161         from pp:MM!MethodContentPackage
162         to ppp:MM2!MethodContentPackage(
163         name <- pp.name,
164         methodContentElement <- pp.methodContentElement,
165         methodPackages <- pp.methodPackages,
166         categories <- pp.categories
167         )
168 }
169 rule Category{
170         from c:MM!Category
171         to cc:MM2!Category(
172         name<-c.name,
173         description <- c.description,
174         subCategory<- c.subCategory,
175         subCategories<- c.subCategories,
176         categorizedElement <- c.categorizedElement)
177 }
178 rule TaskDefinition{
179         from tu:MM!TaskDefinition
180         to tuu:MM2!TaskDefinition(
181         name <- tu.name,
182         description <- tu.description,
183         performer<-tu.performer,
184         participant<-tu.participant,
185         inputs<-tu.inputs,
186         outputs<-tu.outputs,
187         optionalInputs <- tu.optionalInputs,
188         variabilityBasedOnElement<-tu.variabilityBasedOnElement,
189         variabilityType<-tu.variabilityType)
190 }
191 rule RoleDefinition{
192         from ru:MM!RoleDefinition
193         to ruu:MM2!RoleDefinition(
194         description <- ru.description,
195         name <- ru.name,
196         variabilityType <- ru.variabilityType,
197         variabilityBasedOnElement <- ru.variabilityBasedOnElement)
198 }
199 rule WorkProductDefinition{
200         from wpu:MM!WorkProductDefinition
201         to wpuu:MM2!WorkProductDefinition(
202         description <- wpu.description,
203         name <- wpu.name,
204         variabilityType <- wpu.variabilityType,
205         variabilityBasedOnElement <- wpu.variabilityBasedOnElement)
206 }
```

3.6 Related Work

The literature also reports that there are other proposals for generating transformations using MDE concepts. For instance, the Atlas Model Weaver (AMW) [36] is a tool for establishing relationships (i.e., links) between models. These links are stored in a model called weaving model, which is created conforming to a weaving metamodel. AMW provides a base weaving metamodel enabling to create links between model elements and associations between links. Moreover, the weaving model can then be used as input to automatically generate model transformations. However, AMW does not support complex conditions; particularly, it is not possible to include conditions for matching elements as we need for tailoring process models according to the project context.

Other examples are the Model Transformation Language (MOLA) [77] and Graph Rewriting and Transformation (GREaT) [4] that allow specifying transformation rules through visual mapping patterns. They specify rules and mappings using class diagrams, but considering an environment inspired by activity diagrams. A limitation of MOLA and GREaT is that they need that the users to directly interact with metamodels and class diagrams, which still represents a strong usability restriction for these persons. Therefore, if the users do not count on the required experience for managing these elements, they could neither write nor maintain the transformation code.

Another proposal is MTrans [114] that is a rule-based executable approach for describing model transformations. MTrans allows programmers to write arbitrarily complex transformation programs. The expressiveness and generality of the approach constitutes also its main disadvantage, because the programmers must make sure that their program performs correct transformations and that the underlying rule set is consistent. Moreover, no tool support for consistency checking is available.

Finally, according to Tisi et al. [156], the knowledge for generating transformations is verbose, i.e., a long and complex code is required for generating just a small output transformation. This is mainly the problem with technological approaches for building transformations, since they are always complex.

3.7 Summary and Discussion

Although there exist proposals for generating model transformations, these approaches contain complex syntax and constructs that require advanced knowledge in these approaches. Building appropriate tailoring transformations requires expertise for specifying tailoring rules, making decisions, choosing the right kind of transformation, and also mastering the transformation language syntax and semantics. In this sense, we can envision ensuing potential challenges in manually writing ATL tailoring transformations:

Tailoring Rules: For each variable element identified as part of the organizational software process, there is a rule included in the transformation. For optional process elements,

the rule decides, according to the values in the project context model attributes, if it should be included or not in the adapted software process. For process elements defined with alternatives, the rule decides which of them will be included in the adapted software process. Even though this strategy seems quite clear, translating it into ATL rules is a challenging task.

Programming transformations: Writing transformations is usually a complex activity and requires a transformation language knowledge that is not usually known for process engineers. Moreover, there is a lot of syntactical terminology around transformations. These terms are usually only defined with some specific examples to get a clear understanding [78]. Consequently, writing transformations manually can be error-prone.

Organization-dependent: The tailoring transformation is specific to a set of variable processes and context elements that are part of a particular software process. Therefore, the tailoring transformation is not compatible with other software companies that usually have their own software process and context projects. Therefore, knowledge about writing model transformation cannot be reused across organizations.

Evolution: The organizational software process and organizational context model can evolve over time. Evolution can occur at the model or relationships level; model elements can be added, removed or modified; and changes to the relationship between context elements and variable software process elements may lead to a reorganization of the rules, and thus changes to the transformations [57, 58].

Adoption: There are several factors affecting the adoption of software process tailoring in industry [102]: (a) the complexity, expressiveness and understandability of the notations and languages, (b) the cost of coping with variability for particular development scenarios, (c) the degradation of the expensively-captured software process, and (d) the availability and usability of tool support.

Currently most of small and medium-sized companies in Chile do not invest effort in tailoring their software process at all, probably because it is not possible to ensure usefulness of the obtained tailored software process. We have worked with MDE-based software process tailoring for the last five years and it has proved to be technically feasible. However, writing transformations manually is not a cost-effective strategy, if we seek adoption of MDE-based software process tailoring in the software industry. This problem is also valid for large software companies.

The difficulties of the approach proposed in [64] have been considerably reduced with the tools described in this chapter. Now, neither the process model nor the context model need to be manually defined. However, writing tailoring transformations remains a high challenge, and this difficulty is the main motivation for continuing to improve the support for this process. As part of this improvement effort, the next two chapters of this thesis present:

- a DSL for specifying tailoring rules in a usable manner.
- a HOT that automatically generates tailoring transformations in a way that is transparent for the users.

Chapter 4

Tailoring Rules Specification

This chapter presents a domain-specific language for specifying tailoring rules. The tailoring rules are a set rules for tailoring a software process. The domain-specific language is a decision language that allows process engineers to formally specify tailoring rules as a model.

Section 4.1 introduces a motivation for specifying tailoring rules. Section 4.2 shows an overview of domain-specific languages. Section 4.3 presents the decision language for defining tailoring rules using the notion of modelware. Section 4.4 illustrates the use of the decision language using an example. Section 4.5 shows the graphical environment we have built for defining tailoring rules. Finally, section 4.6 discusses some related work about tailoring rules specification.

4.1 Motivation

As mentioned before, process engineers are in charge of tailoring the organizational software process to make it fit to each project using tailoring rules, among other tasks. Tailoring rules are a set of rules that indicates how to tailor an organizational software process according to a specific project context. The tailoring rules involve three main components: concepts, conditions and conclusions, and they can be specified as text, tables, annotations in the software process or other formats. However, the required knowledge is encapsulated and stored, and not necessarily formally specified.

In Chapter 3, we presented the MDE-based tailoring strategy for generating project-adapted software processes by tailoring an organizational software process, applying a set of tailoring rules defined in a tailoring transformation. The tailoring transformation was implemented in ATL and thus it is feasible to use this strategy for tailoring software processes. However, the tailoring rules still need to be written in ATL, which represents a challenge for process engineers. In order to deal with this challenge, the literature reports different approaches for generating transformation rules without interacting with code, but these approaches require MDE formalisms that process engineers usually do not possess. Therefore, if the users do not count on the required experience for managing MDE concepts and pro-

gramming transformations, they can neither implement nor maintain the tailoring rules.

In this sense, we propose a domain-specific language (DSL) for defining tailoring rules without interacting with the ATL code. Using this language the tailoring rules are specified as a model that can be used for generating tailoring transformations. To that end, the DSL is supported by a graphical environment that helps reduce the complexity of defining tailoring rules and specifying tailoring rules as a model.

4.2 An Overview of Domain-specific Languages

Modelware TS enables the definition of models using a domain-specific language (DSL). Executable DSLs hide software implementation details by generating executable code or models. Particularly, domain-specific modeling languages (DSML) –a special type of DSL– are very important within the field of MDE, where each model is written in a specific language, and may be transformed into another model written (in most cases) in yet another language.

According to Greenfield et al. [53], a DSL definition in terms of metamodels (modelware) is more powerful for specifying languages than traditional Backus-Naur Form (BNF) grammars. A DSL definition in the modelware TS considers the following language aspects:

Abstract syntax. Describing the structure of the language and the way the different primitives can be combined together, independently of any particular representation or encoding. The abstract syntax is the definition of the modeling concepts and their properties by defining metamodels which play a role corresponding to grammars for textual languages. While a grammar defines all valid sentences of a language, a metamodel defines all valid models of a modeling language.

Concrete syntax. Describing specific representations of the modeling language, covering encoding and/or visual appearance issues. As mentioned before, metamodels only define the abstract syntax but not the concrete notation of the language, i.e., the graphical or textual elements used to render models elements in modeling editors. The concrete syntax is a representation that the user commonly uses to refer to a particular language.

Static semantics. Describing the meaning of the elements defined in the language and the meaning of the different ways of combining them. The semantics of a language captures the effect of "sentences" of the language. Static semantics expresses the structural meaning of a language term and corresponds to the well-formed rules on top of the abstract syntax (expressed as invariants of metamodel classes).

Additional operations over DSLs. In addition to checked execution by the static semantics, there may be other possible operations manipulating programs written in a given DSL. Each may be defined by a mapping represented by a model transformation.

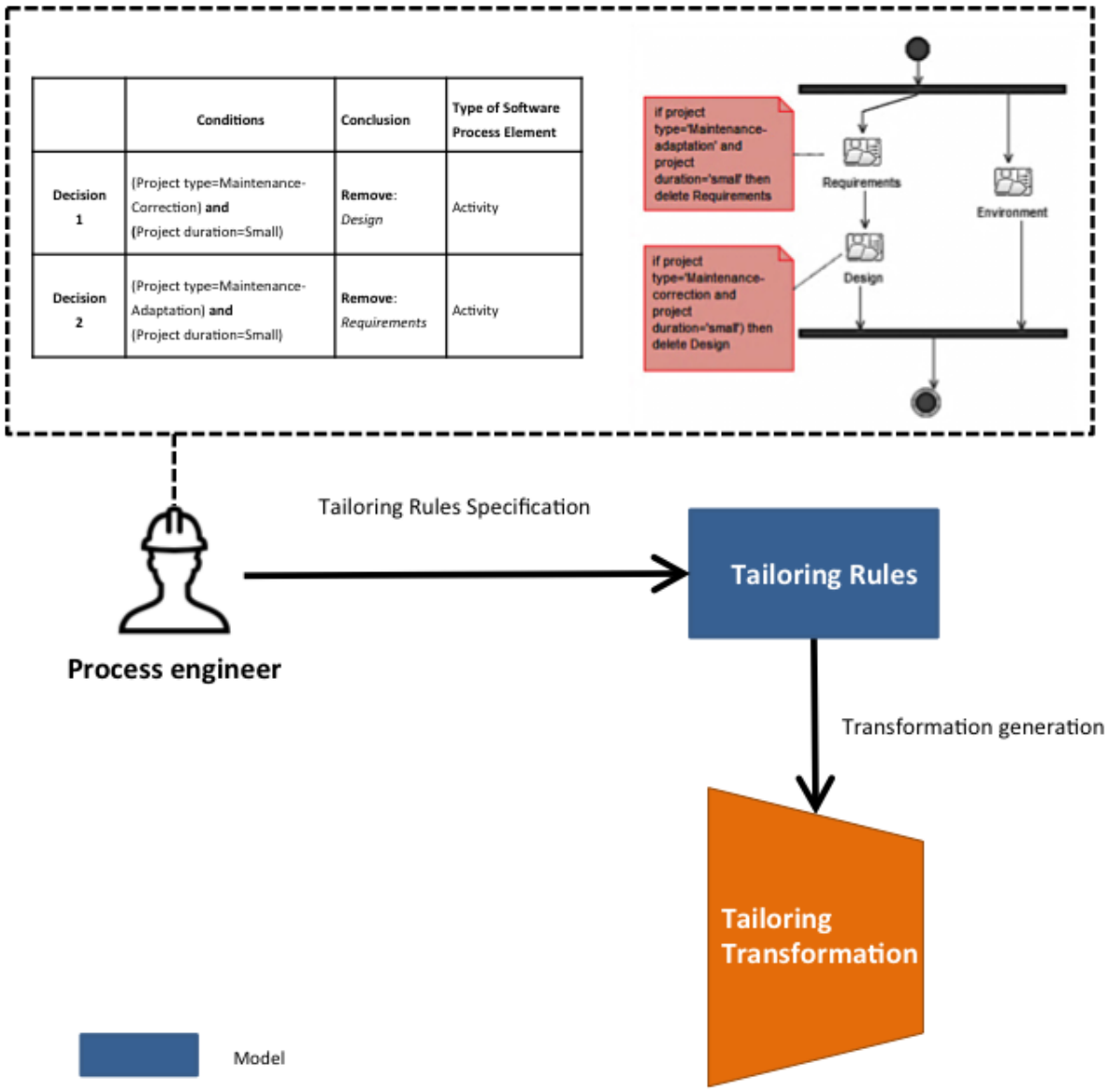


Figure 4.1: An overview of tailoring rules specification.

4.3 Decision Language for Defining Tailoring Rules

In our application domain, the tailoring rules are a set of rules that decide which software process elements to exclude or keep using particular specifications. The upper part of Figure 4.1 shows two kinds of specifications for tailoring rules: tables and annotations. These approaches have some drawbacks: informality, error-proneness and complexity. In order to deal these drawbacks, we propose a *formal specification* of tailoring rules using a domain-specific language.

The decision language (DL) is a domain-specific language that allows users to specify tailoring rules in modelware TS; i.e., the tailoring rules are specified as a model. In this sense, the tailoring rules can be specified in a high-level formalism that is compatible with

the MDE-based tailoring strategy. Consequently, we can use models and transformations for generating tailoring transformations.

The DL definition involves two steps: 1) conceptualizing the domain-specific concepts involved in tailoring rule definition, and 2) formally specifying the DL as a language that allows defining transformation rules.

4.3.1 Conceptualizing the process tailoring rules

A Software Process Line (SPrL) captures the commonalities and variabilities in a set of processes. The set of all the possible software processes that a company follows, i.e., process variants, defines a SPrL. Differences are managed by means of process design decisions, thereby introducing variation points [129]. The whole set of variation points is typically referred to as the variability of the software process line [28].

In the MDE-approach for software process lines [63], tailoring rules are used to resolve on variable process elements. A tailoring rule is a set of rules whose conditions are related to project characteristics (context) and that decide how variable process concepts are resolved when their conditions hold. These rules embody decisions that take place in different process granularity levels (both, optional and alternative).

Process engineers that are in charge of tailoring the software process can specify tailoring rules using natural language. For instance, two tailoring rules are defined for removing two activities, design and requirements, as follows:

"if project type is maintenance-correction and project duration is small, then remove design activity"

"if project type is maintenance-adaptation and project duration is small, then remove requirement activity"

Process engineers can specify tailoring rules using different representations, such as tables, annotations, or text. The upper part of Figure 4.1 shows two tailoring rules for removing design and requirement activities, using tables and annotations. These kinds of representations allow process engineers to identity variable process elements, conditions and conclusions for generating variants of a software process.

Tailoring rules are specified using a particular language in a software process domain. The particular language involves variable process elements, rules and context characteristics. Table 4.1 shows the domain concepts involved in tailoring rules for software process tailoring: variable process elements, rules and context.

Process elements can be either mandatory or variable. Mandatory elements are included in all software processes. Variable elements can be optional or alternative. Optional elements can be either included or not in any software process. Alternative elements have a set of alternatives that should replace the alternative element in all software processes.

Table 4.1: Domain concepts.

	Type	Elements	Definition
Variable process element	Optional or Alternative	Capability pattern	Capability patterns are a special type of process that describes a reusable cluster of activities in common process areas.
		Activity	An activity is a collection of work breakdown elements such as tasks, roles, work products, and milestones. Activities can include other activities. The three activity types are Iterations, Phases and Activities.
		Task	A task is an assignable unit of work. Every task is assigned to a specific role.
Rule	Simple or Composed	Condition	A condition is a logical statement in terms of context attributes and their values, e.g., if project type is maintenance.
		Logical connector	A logical connector is a symbol used to connect simple conditions. These symbols can be operators or quantifiers.
		Conclusion	A conclusion is the statement that "follows" from the condition, e.g., if project type is maintenance then remove requirements.
Context	N/A	Context Attribute	A context attribute is a project characteristic that affects the project context definition, e.g., project size, project duration or team size.
		Context Attribute//Value	A context attribute value is a potential value that defines the context attribute, e.g., project type is a context attribute and it can have the following values: new development, maintenance, incidents or bugs.

The rules are a set of conditions and conclusions that decide on variable process elements. Conditions are a set of predicates for software process tailoring that are stated in terms of simple or compound relationships between context attributes and their values. Conditions can be composed by logical predicates. Conclusions decide actions about concepts that are variable process elements.

4.3.2 Developing the decision language

The DL definition considers three main components [165]: (1) the abstract syntax, (2) the semantics, and (3) the concrete syntax. These components are here specified on the notion of modelware.

4.3.2.1 Abstract syntax

One of the most important aspects of creating a new modeling language is the definition of its abstract syntax. The abstract syntax of a language consists of its set of concepts and respective relationships [18] [165], which together establish a structure for the domain's

information that is relevant to model. Concepts define the domain information that can be stored by models, and relationships define how concepts are related among themselves.

The abstract syntax involves two definitions: the abstract syntax model and the abstract form. The abstract syntax model of a language is a metamodel whose nodes represent the concepts in the language and whose edges represent relationships between these concepts. Abstract form of a language is a labeled graph typed over the abstract syntax model of that language.

Variation Decision Metamodel. The DL's abstract syntax model is specified as a metamodel called Variation Decision Metamodel (VDMM) that describes domain concepts for specifying tailoring rules. The VDMM is inspired by Decision Models [169] and Semantics of Business Vocabulary and Business Rules (SMVB) used for building decision rules [54].

VDMM uses the domain concepts for specifying tailoring rules in terms of process variable elements, rules and contexts. The Fig. 4.2 shows the VDMM that describes the language concepts. *DecisionModel* is the name of the metamodel that considers two main concepts: *ConfigurationContent* and *ConfigurationRule*. These concepts are explained next.

- *ConfigurationContent* incorporates specific information needed for modeling decisions, and particularly it considers *ContextElements* and *VariabilityPointElements*.
  - *ContextElements* is a collection of organizational context elements that includes attributes and their values for a specific context. This collection describes one possible *ContextAttributeValue* for each *ContextAttribute*.
  - *ContextAttribute* represents a relevant characteristic of the project context required for software process tailoring, and it can take one value from a set defined as *ContextAttributeValue*.
  - *ContextAttributeValue* represents a particular value for qualifying a *ContextAttribute*.
  - *VariabilityPointElements* is a set of elements that can be optional (*OptionalPoint*) or alternative (*AlternativePoint*).
  - *OptionalPoint* is a set of software process elements that can be removed for a specific project.
  - *AlternativePoint* is a set of software process elements that can be replaced for other software process element.
- *ConfigurationRule* defines the tailoring rules using domain concepts and it considers a set of *Rules*.
  - *Rule* is a set of *Conditions* and *Conclusions* for each *VariabilityPointElement*.
  - *Conditions* may be simple (include just one condition) using *ConditionSimple*, or complex (consider several conditions with logical connectors) using *ConditionElement* that includes a set of *ConditionSimple* and *LogicalConnector*.
  - *LogicalConnector* is a symbol used to connect a set of *Conditions*.
  - *Conclusion* decides actions about a *Conditions* (simple or complex) for each *OptionalPoint* and *AlternativePoint*. In this sense, the *Conclusion* can be *OptionalConclusion* or *AlternativeConclusion*.

- *OptionalConclusion* specifies the decision for each *OptionalPoint*. The *OptionalConclusion* considers two boolean values: true for keeping an *OptionalPoint* and false for removing an *OptionalPoint*.
- *AlternativeConclusion* specifies the decision for each *AlternativePoint*. The *AlternativeConclusion* is used for replacing an *AlternativePoint* with another software process element.

Variation Decision Metamodel Form. The DL's abstract form is an instance, as a model of VDMM, that contains the relevant information of the linguistic utterance. A linguistic utterance can be defined in the abstract syntax model using labels and constraints between metaclasses, and their relationships.

In this sense, the VDMM shown in Fig. 4.2 contains a conceptual model of the DL and a large number of constraints between metaclasses that allow defining a decision model. The *DecisionModel* is the principal metaclass for deriving a decision model using relationships and constraints between metaclasses of the VDMM. For instance, it ensures that when the model is derived from *DecisionModel* type, the parameters type of an actual type fit the declared formal parameters of the *DecisionModel* type. The description of the constraints is the following:

- *DecisionModel* should define two components: *ConfigurationRule* and *ConfigurationContent* using *myConfigurationRule* and *myConfigurationContent* labels, respectively. The *DecisionModel* only can define one *ConfigurationRule* and one *ConfigurationContent*.
- *ConfigurationRule* defines *Rule* using *myRule* label. The *ConfigurationRule* can be composed by a larger number of *Rules*.
- *Rule* defines *Conditions* and *Conclusion* using *myConditionRule* and *myConclusionRule* labels, respectively. The *Rule* can define a set of *Conditions* and one *Conclusion*.
- *Conditions* defines *ConditionElement* using *myConditionElement* label.
- *Conclusion* defines *OptionalConclusion* and *AlternativeConclusion* using *myOptionalConclusion* and *myAlternativeConclusion* labels, respectively.
- *ConditionElement* defines *ConditionSimple* and *LogicalConnector* for simple conditions, but *ConditionElement* can be defined as a composed condition in terms of a set of *ConditionSimple* using itself for defining another *ConditionElement*.
- *ConfigurationContent* should define *ContextElements* and *VariabilityPointElements* using *myContentContextElements* and *myVariabilityPointElements* labels, respectively.
- *VariabilityPointElements* should be defined as a larger number of *AlternativePoint* and/or *OptionalPoint* using *myAlternativeElement* and *myOptionalElement*, respectively.
- *AlternativePoint* should include one or more *Alternatives* using *myAlternatives*.

According to the literature on decision models [169], conditions should have a left and a right hand side, that in our case are called *ContextAttribute* (left) and *ContextAttributeValue* (right). Similarly, the conclusion must also have a left and a right hand side; in our case they are *VariabilityPointElements* and *myVariabilityValue*, respectively.

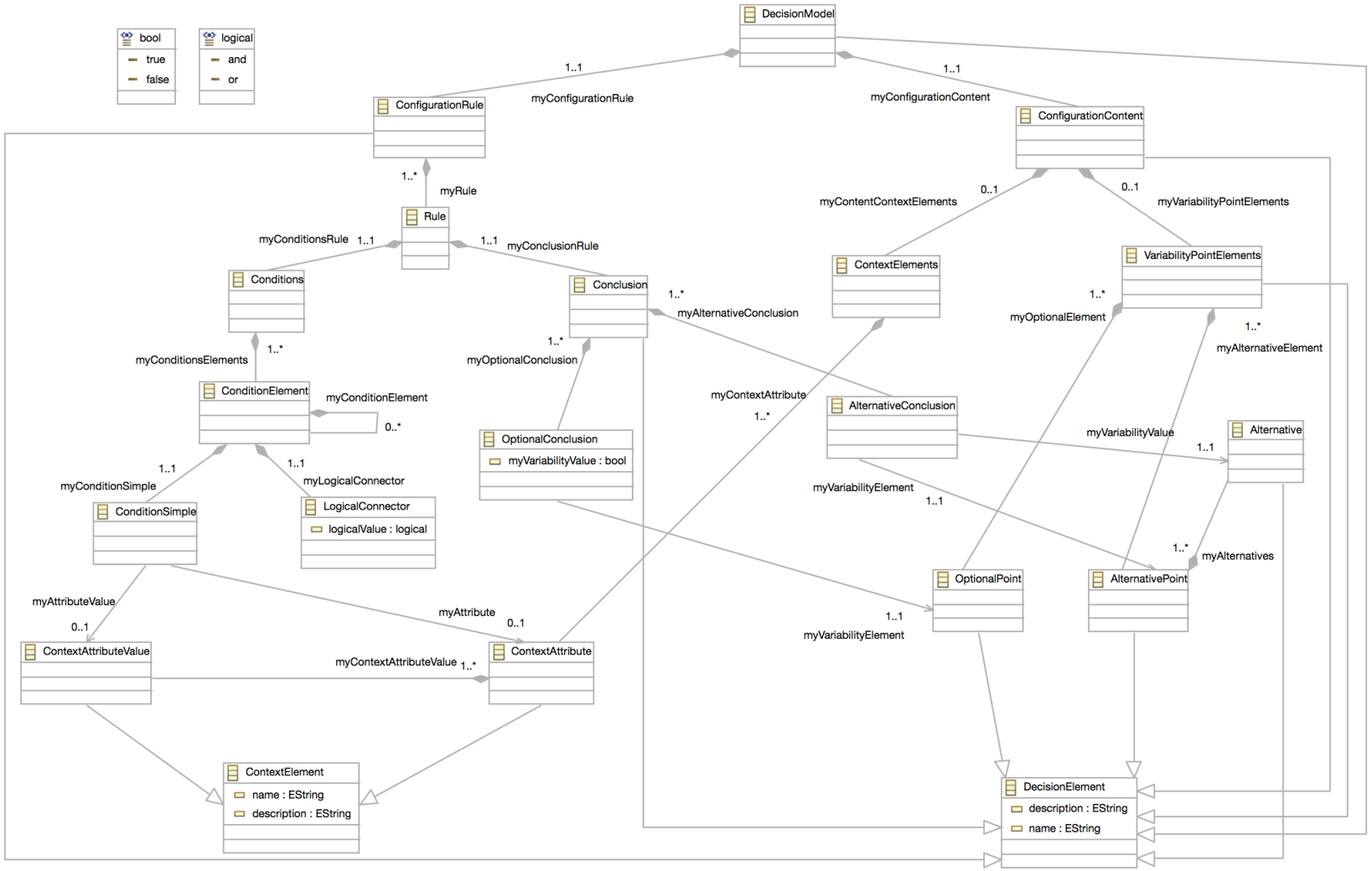


Figure 4.2: Abstract Syntax: Variation Decision Metamodel.

4.3.2.2 Semantics

Semantics describes the meaning of each element and of each orchestration of elements, such as sentences, diagrams, or compositions of modeling elements that constitutes a model. Although technology has not yet evolved to a point in which we can make a computer "understand" a model (namely to ensure its correctness), semantics can nevertheless still be addressed when creating a metamodel. This is done by specifying rules that restrict the different possibilities for element orchestrations in a model, in order to prevent model designers from using invalid ones.

Table 4.2 shows the semantics of the DL. It describes language elements and considers the two main parts of the abstract syntax: *ConfigurationContent* and *ConfigurationRule*. The *ConfigurationContent* considers context attributes and their values, and variable process elements. The *ConfigurationRule* considers the set of rules to adapt the organizational software process, which are defined using the *ConfigurationContent*.

Table 4.2: Semantic: Transformation Rules Language.

Element	**Semantic**	**Action**	**Artifacts of the application domain**
Decision Model	*DecisionModel* is a formal representation of tailoring rules. The decision model considers a particular context for software process tailoring using a set of rules.	DecisionModel derives the tailoring rules and specifying the organizational context.	Organizational software process and Organizational context.
Configuration Rule	*ConfigurationRule* is a specification of tailoring rules. The configuration rules are a set of rules that allow users to define tailoring decisions in terms of conditions and conclusions.	ConfigurationRule is derived from DecisionModel and encapsulates a set of rules.	Organizational software process and Organizational context.
Rule k	*Rule* is a set of conditions and conclusions that decide on variable software process elements. The rules are expressions using organizational context elements for adapting the organizational software process.	Rule decide about variable software process elements.	Organizational software process, Organizational context and Adapted software process.

Table 4.2: Semantic: Transformation Rules Language.

Element	**Semantic**	**Action**	**Artifacts of the application domain**
Condition	*Condition* is a set of predicates for software process tailoring. The condition establishes a relationship between context attributes and their values and can be composed by logical predicates. A condition can be simple or complex.	Condition define a predicate in terms of context attributes and their values for software process tailoring.	Organizational context and Adapted software process.
Condition Element	*ConditionElement* is a expression for defining a simple conditions and logical connectors. Moreover, ConditionElement can be composed for other ConditionElement for defining complex conditions.	ConditionElement define a condition or a logical connector.	Organizational context and Adapted software process
Simple Condition	*SimpleCondition* is a primitive condition. A primitive condition is a expression that is composed by one context attribute and its value.	SimpleCondition define a primitive condition.	Organizational context and Adapted software process
Logical Connector	*LogicalConnector* is a logical operators that allow users to specify complex conditions. The complex conditions establishes two or more relations between simple conditions using the logical operators AND and OR.	LogicalConnector add a logical operator between simple conditions for defining complex conditions.	SimpleCondition and ConditionElement.
Conclusion	*Conclusion* is a expression that considers a set of conditions for deciding about a variable process element(OptionalPoint or AlternativePoint). The conclusion is the tailoring decision for generating the adapted software process	Conclusion define a decision about a variable process element.	Organizational context and Adapted software process

Table 4.2: Semantic: Transformation Rules Language.

Element	**Semantic**	**Action**	**Artifacts of the application domain**
Optional Conclusion	*OptionalConclusion* is a decision for removing an optional process element (OptionalPoint). The OptionalConclusion is a Boolean expression that can be True or False. If the OptionalConclusion is True, the optional process element will not be removed. If the OptionalConclusion is False, the optional process element will be removed.	OptionalConclusion decides about an optional process element.	Organizational software process and Adapted software process.
Alternative Conclusion	*AlternativeConclusion* is a decision for replacing an alternative process element (AlternativePoint). The AlternativeConclusion is an expression that replace a software process element with another software process element.	AlternativeConclusion decides about an alternative process element.	Organizational software process and Adapted software process
Configuration Content	*ConfigurationContent* is a specification of context attributes and variable process elements. The ConfigurationContent consider context and software process elements that are used for defining the ConfigurationRule.	ConfigurationContent is derived from "DecisionModel" and encapsulates context and software process elements.	Organizational context.
Context Element	*ContextElement* is a set of context attributes and their values that are part of organizational context. The ContextElements are used for defining conditions of the tailoring rules.	ContextElement defines context attributes and their potential values.	Organizational context.
Context Attribute	*ContextAttribute* represents a relevant characteristic of the project context required for software process tailoring and can take one of a set of values defined as ContextAttributeValue.	ContextAttribute defines a relevant characteristic of a particular project.	Organizational context.

Table 4.2: Semantic: Transformation Rules Language.

Element	**Semantic**	**Action**	**Artifacts of the application domain**
Context Attribute Value	*ContextAttributeValue* represents a value for qualifying a ContextAttribute. A ContextAttribute can have one or more ContextAttributeValue.	ContextAttributeValue defines the potential values for qualifying a ContextAttribute.	Organizational context.
Variability Point Elements	*VariabilityPointElement* is a set of variable software process elements that are considers for software process tailoring. The variable process elements can be Optional or Alternative.	VariabilityPointElement define variable software process elements.	Organizational software process and Adapted software process.
Optional Point	*OptionalPoint* is an optional process element that can be removed in the adapted software process.	OptionalPoint defines an optional process element.	Organizational software process and Adapted software process.
Alternative Point	*AlternativePoint* is an alternative process element that can be replaced by another process element in the adapted software process.	AlternativePoint defines an alternative process element.	Organizational software process and Adapted software process.
Alternative	*Alternative* is a set of process elements that can replace an AlternativePoint in the adapted software process.	Alternative defines process elements for replacing another software process.	Organizational software process and Adapted software process.

4.3.2.3 Concrete syntax

Concrete syntax provides the textual representation of the metamodel elements, which facilitates the presentation and construction of a model. The concrete syntax of the DL for defining tailoring rules has a textual representation based on Extensible Markup Language (XML).

Table 4.3 shows the concrete syntax of the DL. It describes constructors for defining a Decision Model, constraints and dependencies. For example, $< myRule >< \backslash myRule >$ is a construct to describe a *Rule* and it can be used one or more times (constraint).

Table 4.3: Concrete Syntax: Transformation Rules Language.

Element	**Construct**	**Action**	**Constraints**	**Dependencies**
Decision Model	$< DecisionModel >$ $< \backslash DecisionModel >$	New decision model	1 ... 1	
Configuration Rule	$< myConfigurationRule >$ $< \backslash myConfigurationRule >$	New configuration rule	1 ... 1	Decision Model
Rule	$< myRule >$ $< \backslash myRule >$	New rule	1 ... *	Configuration Rule
Condition	$< myConditionRule >$ $< \backslash myConditionRule >$	New condition	1 ... 1	Rule
Condition Element	$< myConditionElements >$ $< \backslash myConditionElements >$	New condition element that support composed condition	0 ... *	Condition Condition Element
Simple Condition	$< mySimpleCondition >$ $< \backslash mySimpleCondition >$	New simple condition	1 ... 1	Condition Element
Logical Connector	$< myLogicalConnector >$ $< \backslash myLogicalConnector >$	Add a logical connector between simple condition and other simple conditions or between condition elements	1 ... 1	Condition Element
Conclusion	$< myConclusionRule >$ $< \backslash myConclusionRule >$	New conclusion	1 ... 1	Rule
Optional Conclusion	$< myOptionalConclusion >$ $< \backslash myOptionalConclusion >$	New optional conclusion	1 .. *	Conclusion
Alternative Conclusion	$< myAlternativeConclusion >$ $< \backslash myAlternativeConclusion >$	New alternative conclusion	1 ... *	Conclusion
Configuration Content	$< myConfigurationContent >$ $< \backslash myConfigurationContent >$	New configuration content	1 ... 1	Decision Model
Context Element	$< myContentContextElement >$ $< \backslash myContentContextElement >$	New context element	0 ... 1	Configuration Content
Context Attribute	$< myContextAttribute >$ $< \backslash myContextAttribute >$	New context attribute	1 ... 0	Context Element
Context Attribute Value	$< myAttributeValue >$ $< \backslash myAttributeValue >$	New context value	1 ... *	Context Attribute
Variability Point Elements	$< myVariabilityPointElements >$ $< \backslash myVariabilityPointElements >$	New variability point elements	0 ... 1	Configuration Content
Optional Point	$< myOptionalElement >$ $< \backslash myOptionalElement >$	List of optional points	1 ... *	Variability Point Elements
Alternative Point	$< myAlternativeElement >$ $< \backslash myAlternativeElement >$	List of alternative points	1 ... *	Variability Point Elements
Alternative	$< myAlternatives >$ $< \backslash myAlternatives >$	List of alternatives	1 ... *	Alternative Point

A textual representation of the DL for defining tailoring rules is presented in listing 4.1. The textual representation of the DL considers three kinds of sentences: (1) Constructs (black in listing 4.1), (2) Attributes (blue in listing 4.1) and Attribute Values (red in listing 4.1).

Listing 4.1: Concrete syntax: Textual representation of the DSL.

```
1  //Concrete Syntax of the DL
2  <DecisionModel>
3  //Configuration Content: Context Elments and Variability Points
4  <myConfigurationContent xmi:id="myConfCont">
5          //Context Elements and its values
6          <myContentContextElements xmi:id="myContConteEle">
7          <myContextAttribute name="Project Type" xmi:id="Project Type">
8          <myContextAttributeValue name="New Project" xmi:id="New Project"/>
9          <myContextAttributeValue name="Maintenance" xmi:id="Maintenance"/>
10         </myContextAttribute>
11         <myContextAttribute name="Team Size" xmi:id="Team Size">
12         <myContextAttributeValue name="Small" xmi:id="Small"/>
13         <myContextAttributeValue name="Medium" xmi:id="Medium"/>
14         <myContextAttributeValue name="Large" xmi:id="Large"/>
15         </myContextAttribute>
16         <myContextAttribute name="Project Duration" xmi:id="Project Duration">
17         <myContextAttributeValue name="Small" xmi:id="Small"/>
18         <myContextAttributeValue name="Medium" xmi:id="Medium"/>
19         <myContextAttributeValue name="Large" xmi:id="Large"/>
20         </myContextAttribute>
21         </myContentContextElements>
22         //Optional and alternative points
23         <myVariabilityPointElements xmi:id="_gXraUPgVEeKIEco4bOm3fg">
24         <myOptionalElement name="Requirements" xmi:id="Requirements"/>
25         <myOptionalElement name="Use Cases" xmi:id="Use Cases"/>
26         <myOptionalElement name="Design" xmi:id="Design"/>
27         <myAlternativeElement xmi:id="Establish Requirements Baseline" name="Establish Requirements
               Baseline">
28         <myAlternatives xmi:id="Establish Requirements Baseline without Test Cases" name="Establish
               Requirements Baseline without Test Cases"/>
29         <myAlternatives xmi:id="Establish Requirements Baseline and Test Cases" name="Establish
               Requirements Baseline and Test Cases"/>
30         </myAlternativeElement>
31         </myVariabilityPointElements>
32 </myConfigurationContent>
33
34 //Configuration Rules: Rules and its conditions
35 <myConfigurationRule>
36         //Rule 1 with optional conclusion
37         <myRule>
38         <myConditionsRule>
39         <myConditionsElements>
40         <myConditionSimple myAttribute="Project Type" myAttributeValue="Maintenance"/>
41         </myConditionsElements>
42         </myConditionsRule>
43         <myConclusionRule>
44         <myOptionalConclusion myVariabilityElement="Requirements" myVariabilityValue="false"/>
45         </myConclusionRule>
46         </myRule>
47         //Rule 2 with alternative conclusion
48         <myRule>
49         <myConditionsRule>
50         <myConditionsElements>
51         <myConditionSimple myAttribute="Team Size" myAttributeValue="Small"/>
52         </myConditionsElements>
53         <myConditionsElements>
54         <myLogicalConnector logicalValue="or"/>
55         </myConditionsElements>
56         <myConditionsElements>
57         <myConditionSimple myAttribute="Project Duration" myAttributeValue="Small"/>
58         </myConditionsElements>
59         </myConditionsRule>
60         <myConclusionRule>
61         <myAlternativeConclusion myVariabilityElement="Establish Requirements Baseline"
               myVariabilityValue="Establish Requirements Baseline without Test Cases"/>
62         </myConclusionRule>
63         </myRule>
64 </myConfigurationRule>
65 </DecisionModel>
```

The concrete syntax of the DL for defining tailoring rules considers the abstract syntax to specify *myConfigurationContent* (context elements and variability points) and *myConfigurationRules* (rules, conditions and conclusions).

4.4 Using the Decision Language

In Chapter 3, we defined a running example to illustrate the MDE-based tailoring strategy. The running example considers part of Rhiscom's software process and describes decisions for four variable process elements. Two decisions are defined for removing two optional process elements: *Design* and *Requirements*; and two other decisions are defined for replacing two alternative process elements: *Specify requirements* and *Establish requirements baseline*.

We use our decision language (DL) for specifying these rules as a Variation Decision Model (VDM). The VDM is a model that is defined using our DL. In this sense, we need to apply two steps: (1) Identify the domain concepts of tailoring decision, (2) Define a VDM using the DL.

4.4.1 Identifying the domain concepts

Rhiscom defined tailoring decisions in natural language. In this sense, we need to identify and specify the domain concepts of these tailoring decisions.

Decision 1: "if project type is maintenance-adaptation and project duration is small, then remove Requirements activity".

This tailoring rule is in natural language and its specification in terms of domain concepts is the following:

Variable element:
Optional: Requirements activity
Rule:
Condition: project type is maintenance-adaptation
Logical predicate: and
Condition: project duration is small
Conclusion: Remove requirements
Context:
Context attribute: project type
Context attribute value: maintenance-adaptation
Context attribute: project duration
Context attribute value: small

Decision 2: "if project type is maintenance-correction and project duration is small, then remove Design activity".

Now the specification in terms of domain concepts is the following:

Variable element:
Optional: Design activity
Rule:

Condition: project type is maintenance-correction
Logical predicate: and
Condition: project duration is small
Conclusion: Remove design
Context:
Context attribute: project type
Context attribute value: maintenance-correction
Context attribute: project duration
Context attribute value: small

Decision 3: "if (project type is incidents or project type is maintenance-enhancement) and business knowledge is known, then replace Specify requirements by Specify requirements in plain text".

The specification in terms of domain concepts is the following:

Variable element:
Alternative: Specify requirements
Rule:
Condition: project type is incidents or project type is maintenance-enhancement
Logical predicate: and
Condition: business knowledge is known
Conclusion: Replace specify requirements by specify requirements in plain text
Context:
Context attribute: project type
Context attribute value: incidents
Context attribute: project type
Context attribute value: maintenance-enhancement
Context attribute: business knowledge
Context attribute value: known

Decision 4: "if project type is new development and project duration is medium and business knowledge is unknown, then replace Establish requirements baseline by Establish requirements baseline and test cases".

The specification in terms of domain concepts is the following:

Variable element:
Alternative: Establish requirements baseline
Rule:
Condition: project type is new development
Logical predicate: and
Condition: project duration is medium
Logical predicate: and
Condition: business knowledge is unknown
Conclusion: Replace Establish requirements baseline by Establish requirements baseline and

test cases
Context:
Context attribute: project type
Context attribute value: new development
Context attribute: project duration
Context attribute value: medium
Context attribute: business knowledge
Context attribute value: unknown

4.4.2 Defining a VDM using the DL

We have identified the domain concepts for each decision. Then, we need to define a VDM. Listing 4.2 shows Rhiscom's VDM that is defined using our DL and its concrete syntax.

Listing 4.2: Rhiscom's VDM using textual representation of the DL.

```
1  //Concrete Syntax of the DL
2  <DecisionModel>
3  //Configuration Content: Context Elments and Variability Points
4  <myConfigurationContent xmi:id="myConfCont">
5  //Context Elements and its values
6  <myContentContextElements xmi:id="myContConteEle">
7  <myContextAttribute name="Project Type" xmi:id="Project type">
8  <myContextAttributeValue name="New Project" xmi:id="New development"/>
9  <myContextAttributeValue name="Maintenance" xmi:id="Maintenance-adaptation"/>
10 <myContextAttributeValue name="Maintenance" xmi:id="Maintenance-correction"/>
11 <myContextAttributeValue name="Maintenance" xmi:id="Maintenance-enhancement"/>
12 <myContextAttributeValue name="Maintenance" xmi:id="Incidents"/>
13 </myContextAttribute>
14 <myContextAttribute name="Team Size" xmi:id="Project duration">
15 <myContextAttributeValue name="Small" xmi:id="Small"/>
16 <myContextAttributeValue name="Medium" xmi:id="Medium"/>
17 <myContextAttributeValue name="Large" xmi:id="Large"/>
18 </myContextAttribute>
19 <myContextAttribute name="Project Duration" xmi:id="Business knowledge">
20 <myContextAttributeValue name="Small" xmi:id="Known"/>
21 <myContextAttributeValue name="Medium" xmi:id="Affordable"/>
22 <myContextAttributeValue name="Large" xmi:id="Unknown"/>
23 </myContextAttribute>
24 </myContentContextElements>
25 //Optional and alternative points
26 <myVariabilityPointElements xmi:id="_gXraUPgVEeKIEco4bOm3fg">
27 <myOptionalElement name="Requirements" xmi:id="Requirements"/>
28 <myOptionalElement name="Design" xmi:id="Design"/>
29
30 <myAlternativeElement name="Specify Requirements" xmi:id="Specify Requirements"/>
31 <myAlternatives xmi:id="Specify Requirements in plain text" name="Specify Requirements in plain text"/>
32 <myAlternatives xmi:id="Specify Requirements in use cases" name="Specify Requirements in use cases"/>
33 </myAlternativeElement>
34 <myAlternativeElement xmi:id="Establish Requirements Baseline" name="Establish Requirements Baseline">
35 <myAlternatives xmi:id="Establish Requirements Baseline without Test Cases" name="Establish Requirements
       Baseline without Test Cases"/>
36 <myAlternatives xmi:id="Establish Requirements Baseline and Test Cases" name="Establish Requirements
       Baseline and Test Cases"/>
37 </myAlternativeElement>
38 </myVariabilityPointElements>
39 </myConfigurationContent>
40
41 //Configuration Rules: Rules and its conditions
42 <myConfigurationRule>
43 //Rule 1 with optional conclusion
44 <myRule>
45 <myConditionsRule>
46 <myConditionsElements>
47 <myConditionSimple myAttribute="Project type" myAttributeValue="Maintenance-adaptation"/>
48 <myConditionsElements>
49 <myLogicalConnector logicalValue="or"/>
50 </myConditionsElements>
51 <myConditionsElements>
52 <myConditionSimple myAttribute="Project duration" myAttributeValue="Small"/>
53 <myConditionsElements>
54 </myConditionsRule>
55 <myConclusionRule>
56 <myOptionalConclusion myVariabilityElement="Requirements" myVariabilityValue="false"/>
57 </myConclusionRule>
58 </myRule>
59 //Rule 2 with optional conclusion
```

```
<myRule>
<myConditionsRule>
<myConditionsElements>
<myConditionSimple myAttribute="Project type" myAttributeValue="Maintenance-correction"/>
</myConditionsElements>
<myConditionsElements>
<myLogicalConnector logicalValue="and"/>
</myConditionsElements>
<myConditionsElements>
<myConditionSimple myAttribute="Project Duration" myAttributeValue="Small"/>
</myConditionsElements>
</myConditionsRule>
<myConclusionRule>
<myOptionalConclusion myVariabilityElement="Design" myVariabilityValue="false"/>
</myConclusionRule>
</myRule>
//Rule 3 with alternative conclusion
<myRule>
<myConditionsRule>
<myConditionsElements>
<myConditionSimple myAttribute="Project type" myAttributeValue="incidents"/>
</myConditionsElements>
<myConditionsElements>
<myLogicalConnector logicalValue="or"/>
</myConditionsElements>
<myConditionsElements>
<myConditionSimple myAttribute="Project type" myAttributeValue="Maintenance-enhancement"/>
</myConditionsElements>
<myLogicalConnector logicalValue="and"/>
</myConditionsElements>
<myConditionSimple myAttribute="Business knowledge" myAttributeValue="known"/>
</myConditionsElements>
</myConditionsRule>
<myConclusionRule>
<myAlternativeConclusion myVariabilityElement="Specify Requirements" myVariabilityValue="Specify
    Requirements in plain text"/>
</myConclusionRule>
</myRule>
//Rule 4 with alternative conclusion
<myRule>
<myConditionsRule>
<myConditionsElements>
<myConditionSimple myAttribute="Project type" myAttributeValue="New development"/>
</myConditionsElements>
<myConditionsElements>
<myLogicalConnector logicalValue="and"/>
</myConditionsElements>
<myConditionsElements>
<myConditionSimple myAttribute="Project duration" myAttributeValue="Medium"/>
<myConditionsElements>
<myLogicalConnector logicalValue="and"/>
</myConditionsElements>
<myConditionSimple myAttribute="Business knowledge" myAttributeValue="Unknown"/>
</myConditionsElements>
</myConditionsRule>
<myConclusionRule>
<myAlternativeConclusion myVariabilityElement="Establish Requirements Baseline" myVariabilityValue="
    Establish Requirements Baseline and Test Cases"/>
</myConclusionRule>
</myRule>
</myConfigurationRule>
</DecisionModel>
```

Rhiscom's VDM can be shown using Eclipse Modeling Framework (EMF). Figure 4.3 shows a visual representation of Rhiscom's VDM as defined in listing 4.2. In Rhiscom's VDM, we can see context elements, variability points and rules. Context elements include three context attributes and their values: Project type, project duration and Business knowledge. Variability points includes four software process elements: two optional (Design and Requirements) and two alternative (Specify Requirements and Establish Requirements Baseline). Rules includes four tailoring rules with their conditions and conclusions. Finally, in order to visualize a tailoring rule specification, rule 4 is highlighted to show its conditions and conclusion in the properties section that provides EMF.

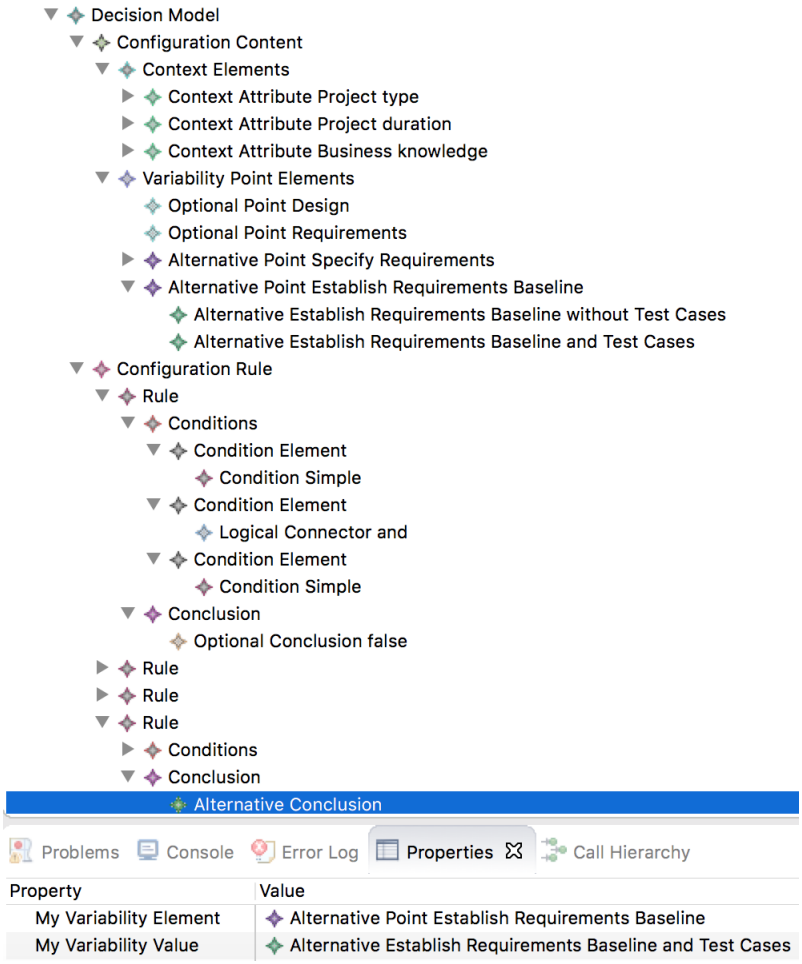


Figure 4.3: Rhiscom's VDM from eclipse modeling framework.

4.5 Graphical Environment for Defining Tailoring Rules

In order to improve the usability of our DL that allow defining a VDM, we developed a graphical environment for defining tailoring rules. The process engineer uses the graphical environment to indicate the organizational software process and organizational context that will be used in the definition of tailoring rules. After that, she/he can define tailoring rules for each variable software process element.

The graphical environment has three steps: (1) selection of the organizational software process and organizational context, (2) selection of variable software process elements, and (3) definition of tailoring rules using our DL that is supported by a user interface.

In the first step, the process engineer uses an interface to select the organizational software process with variability that is exported from EPF as XML file, and the organizational context that is generated from context modeling tool as XML file. Figure 4.4 shows a user interface for selecting the organizational software process with variability and the organizational context of Rhiscom software company. Moreover, the organizational software process is transformed in the organizational software process model as XMI file using our *Injector* (see Chapter 2)

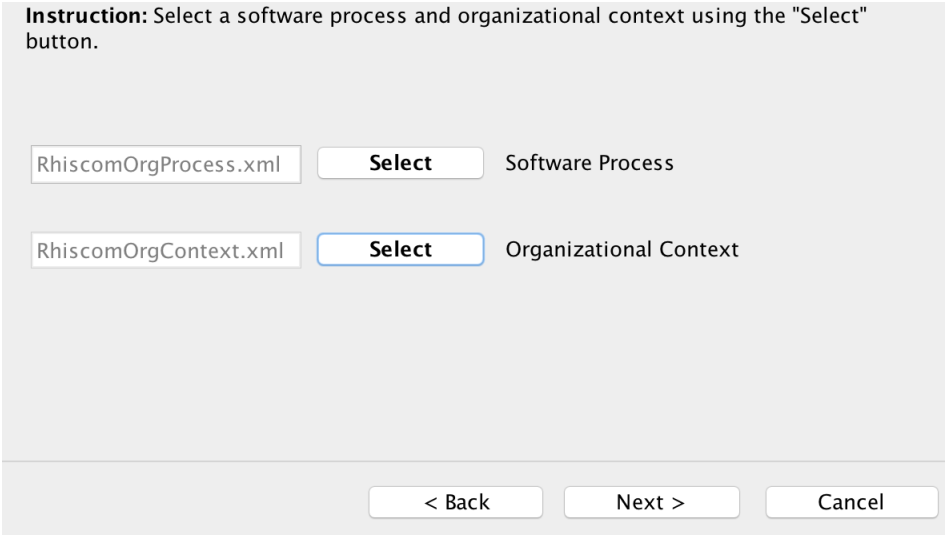


Figure 4.4: Selection of the organizational software process with variability and the organizational context.

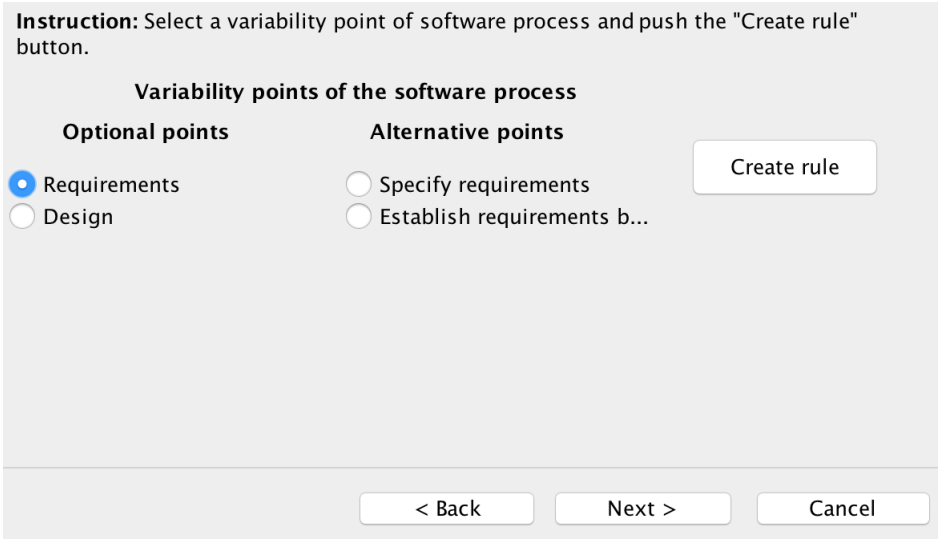


Figure 4.5: Selection of variable software process elements.

that can be used for MDE-based software process tailoring.

In the second step, the user interface presents a set of variable software process elements that can be selected from the organizational software process. The variable process elements can be either optional or alternative. Then, the process engineer uses the graphical environment to indicate which variable process element is selected for defining its tailoring rules. Figure 4.5 shows two optional variation points for Rhiscom's software process: *Requirements* and *Design*, and also two alternative variation points: *Specify Requirements* and *Establish Requirements Baseline*. If the user selects a variation point (e.g., *Requirements*) and clicks on the "Create Rule" button, she/he can define the rules that will be used to tailor the organizational process in such a point, depending on the values of the context attributes of a specific project.

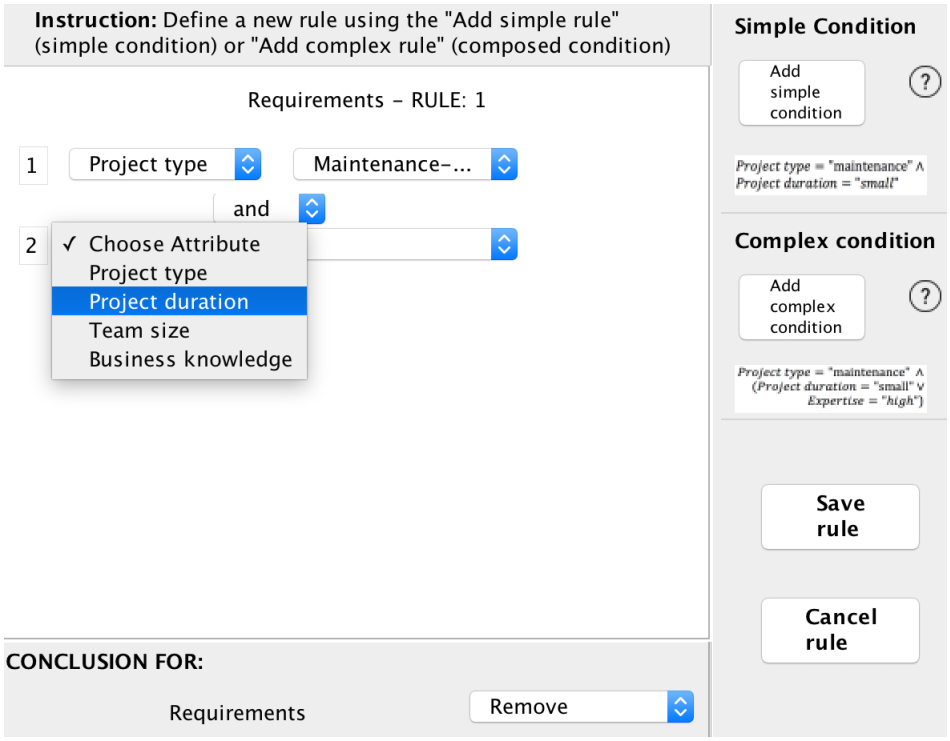


Figure 4.6: User interface for defining tailoring rules.

In the third step, the process engineer interactively defines the relationships between the context attribute values and the variable process elements. The graphical interface supports the DL for defining tailoring rules and allows specifying a VDM. Figure 4.6 shows a tailoring rule definition using our DL and the graphical environment. The process engineer defines that the *Requirements* activity should be removed when the *Project Type* is *Maintenance-adaptation* and, *Project Duration* is *Small*. This decision is part of the adaptation rules defined by Rhiscom for its organizational software process.

Finally, the graphical environment generates a formal specification of the tailoring rules as a VDM using our DL. The models generated from our graphical environment can be imported into EMF. Moreover, the main purpose of building a graphical environment is to aid the process engineer in using our DL without interacting with the concrete syntax.

4.6 Related Work

Proposals trying to address MDE solutions in general should balance the formality required by MDE and the usability needed for final users, if the solution is intended for industrial applications. As mentioned before, proposals like Model Transformation Language (MOLA) [77] and Graph Rewriting and Transformation (GREaT) [4] allow specifying transformation rules through visual mapping patterns, that allow establishing relationships between metamodel attributes and elements from the source and target models. In these language the users need to directly interact with metamodels and class diagrams, which reduces the usability and adoption of these representations (at least for these end-users). therefore, users without the

required experience become unable to write or maintain the code of the transformations.

Varró and Balogh propose the Visual Automated Model Transformations (VIATRA) framework [162] that provides a rule editor (in textual format) for specifying patterns. This tool is supported by Eclipse, but it has the same usability limitations as the previous tools, since VIATRA does not have a graphical environment for defining rules and establishing its own syntax for defining and executing the resulting transformations.

Other option is AToMPM (A Tool for Multi-Paradigm Modeling) [153] that is a Web-based metamodeling and transformation tool for multi-paradigm modeling. It allows defining DSLs through an interactive interface for defining its abstract syntax. It also provides support for rule definition and scheduling using graph transformation rules. In this sense their approach is similar to the one proposed in this thesis. However, in AToMPM rules still need to be specified using a custom language, and they only support simple conditions.

Some researchers have developed domain-specific languages as well as tools that support the rule specification; for instance, Jia et al. [70] developed ZOOM, a platform for its own DSL that allows textual specification of rules, and whose main advantage is the possibility of inserting pre and post-conditions for analysis purposes. Sijtema [140] proposes an extension to the Atlas Transformation Language (ATL), which is based on feature models and requires the specification of the so-called variability rules. In this approach, the variability rules are transformed into standard ATL code, i.e., variability is translated to *called rules* in ATL using a HOT. Although the proposal raises the abstraction level, the process engineer still needs to understand the ATL extension and interact with the source code.

4.7 Summary and Discussion

The literature reports DSLs for defining transformation rules in different abstraction levels, using semi-automatic generators and visual tools. This review provided a basis and useful ideas for implementing and developing a DSL for defining transformation rules in the software process domain.

To that end, we conceptualized the process tailoring rules in terms of domain concepts and the developed decision language (DL) for defining tailoring rules. The domain concepts are a set of keywords that software process engineers use for specifying tailoring rules.

By keeping in mind the goal of reducing the skills required to define tailoring rules, we developed a graphical user interface that allows software process engineer to use the DL for building decision models. In this sense, the DL allows users to generate a variation decision model (VDM) that is a formal representation of tailoring rules as a model (i.e., a VDM that conforms to VDMM). This interface allows the users to define tailoring rules for each variation point of the organizational software process to be tailored without interacting with the concrete syntax of the DL.

This activity involves two steps: the definition of tailoring rules and the automatic generation of the decision model (VDM). Concerning the first one, the tool guides the users

presenting process elements defined as variables, and allows them to specify a rule (either simple or complex) for each variation point. The tool also counts on a menu where the context attributes and their potential values can be selected, not requiring any typing. The second step is done without the user intervention.

The proposed graphical user interface allows conducting a more usable, useful and less error-prone procedure for defining tailoring rules. So far, we have not found any tailoring rule that could not be specified using the proposed DL, raising our confidence about its expressiveness for this application domain. However, more practical experimentation is required to have conclusive evidence.

Chapter 5

Tailoring Transformation Generation

This chapter presents a higher-order transformation (HOT) and an ATL extractor for generating tailoring transformations. The HOT is a model-to-model transformation that takes the variation decision model as input and generates a transformation model as output. The ATL extractor is a model-to-text transformation that takes a transformation model and generates a tailoring transformation. The tailoring transformation is generated in an automatic manner and it can be used to perform software process tailoring.

Section 5.1 introduces a motivation for generating tailoring transformations. Section 5.2 shows an overview of higher-order transformations. Section 5.3 defines the higher-order transformation for generating transformation models. Section 5.4 presents the ATL extractor for generating transformation code from a transformation model. Section 5.5 illustrates the use of the higher-order transformation and ATL extractor using an example. Section 5.6 presents a tool for the configuration and execution of higher-order transformation and ATL extractor. Finally, section 5.7 discusses the related work concerning transformation generation.

5.1 Motivation

In the MDE-based tailoring approach proposed by Hurtado et al. [65], a tailoring transformation is usually programmed in ATL, and it is supported by tools developed in Eclipse, which support the use of model transformations. However, programming transformations are complex themselves because expertise is required for describing transformation rules, and for mastering transformation languages.

In Chapter 2, we presented the tailoring transformation that was manually implemented in ATL. Writing tailoring transformations is not an easy task because the process engineer needs to know about the transformation language. Actually, there are some proposals for automatically generating part of the transformations, but they are not easily applicable in the software industry because the potential users usually do not have the skills for developing or maintaining the non-automatically generated part of the transformations.

Provided that transformations can also be considered as models [12], HOTs [156] are special kind of transformations that takes a transformation as input and/or generates a transformation as output. In particular, Atlas Model Weaver (AMW) [36] is a proposal that uses HOTs for generating transformation models using a weaving model that defines the relationships between two models. We follow the structure of the AMW for our solution [142]: defining the relationship between both, input models and the output model, using the DSL defined in the previous chapter and also this model as the input for a HOT to generate a transformation model.

According to Mens et al. [98], the transformation models can range from abstract analysis models over more concrete design models, to very concrete models of source code. In this sense, transformation models involve a kind of ability to reuse, guarantee correctness, and compose and verify the transformations that allow dealing with complete or consistent models. Moreover, transformation models may be transformed into other models until the model can be made executable using a code generator [170]. In particular, the target model of a transformation can be a transformation itself, possibly defined in a different transformation language, such as ATL, QVT or XSLT.

In order to establish a bridge between technical spaces, AMW tool considers a pluggable architecture for enabling an integration of weaving models and ATL transformations. In this sense, AMW tool provides injectors and extractors for generating ATL transformations.

We propose a HOT and an ATL extractor for automatically generating tailoring transformations. The HOT has one input model (the VDM), and generates the tailoring transformation model that conforms to the ATL metamodel. The ATL extractor generates the tailoring transformation from the tailoring transformation model.

5.2 An Overview of Higher-Order Transformation

Generators and transformations allow for the synthesis of different artifacts, such as source code or alternative model representations, and also the transformation of different models with the possibility of ensuring consistency. Transformations and generators can be grouped using different criteria, for example, depending on whether they are imperative or declarative, how many input and output models are involved, if they are horizontal or vertical, whether they directly manipulate a model, if they are described via algebraic relations, are graph-based or use a hybrid approach [32, 98].

These transformation manipulations can be implemented by means of several MDE technologies, like program transformations [156] or multi-paradigm modeling [153]. The MDE paradigm allows the use of the same transformation infrastructure also for transformation manipulation, by considering model transformations as a particular kind of model. The transformation is represented by a transformation model that conforms to a transformation metamodel. Just as any other model, it can be created, modified, and augmented by a transformation. Moreover, a transformation model can itself be instantiated or modified by a so-called Higher-Order Transformation [156].

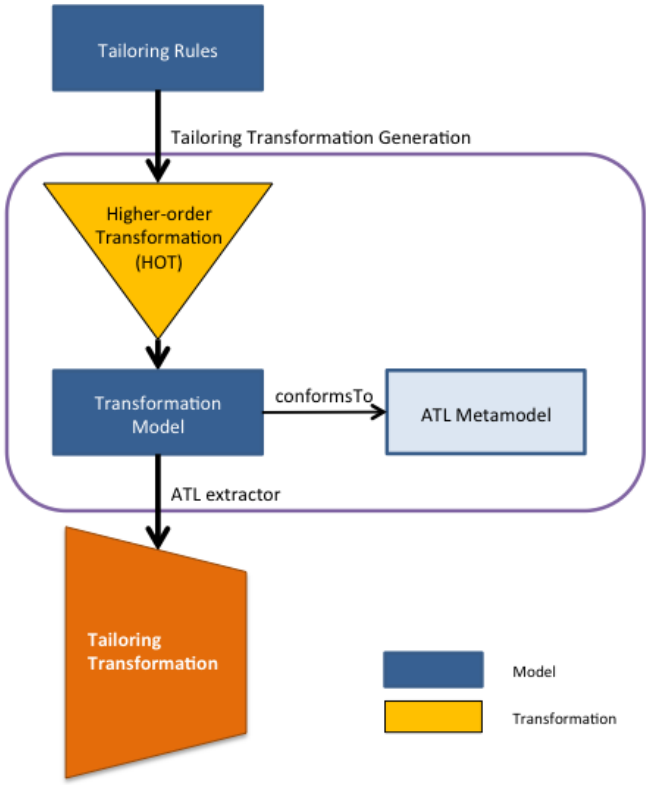


Figure 5.1: An overview of tailoring transformation generation.

HOTs are transformations that take a model as input and generate a transformation as output. They are transformations for modifying/reading/creating model transformations. In the HOT approach transformations must be treated as models conforming to a relevant metamodel. Although the HOT idea can be applied to any transformation language, the most proposed HOTs have been created for the ATL language [156]. A specific use-case for HOTs is the translation of models, where HOTs are developed to translate model elements from a particular model to other models or transformations.

5.3 Higher-Order Transformation for Generating Transformation Models

Considering transformations and generator goals, we propose the *automatic generation* of a tailoring transformation using both a HOT and an extractor. Figure 5.1 shows the structure of our proposed solution for tailoring transformation generation using a HOT and an ATL code extractor. The HOT takes the tailoring rules as a VDM that conforms to VDMM, and it generates a transformation model that conforms to ATL metamodel. The ATL code extractor takes the transformation model and generates tailoring transformation code. This tailoring transformation generation as a whole produces ATL code that can then be used for software process tailoring.

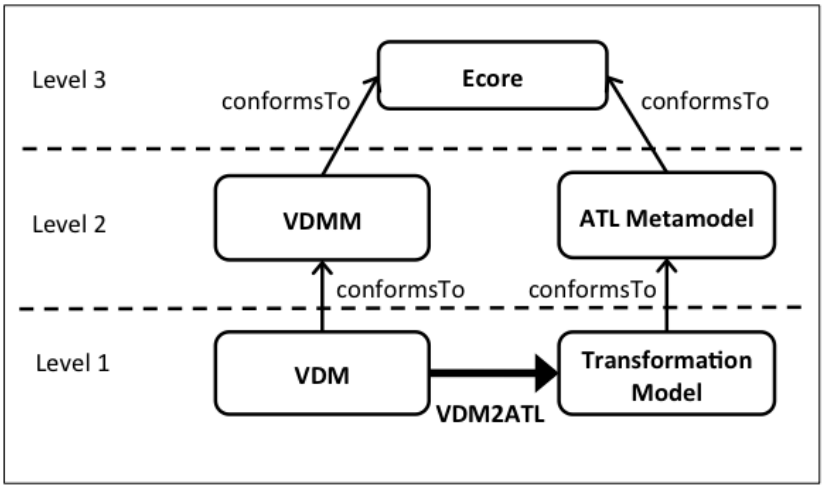


Figure 5.2: A HOT for generating tailoring transformations.

Next we describe the HOT in two steps: 1) implementation of the HOT, and 2) benefits and drawbacks of the HOT.

5.3.1 Implementation

Our HOT implements a transformation synthesis pattern for generating a transformation model using a model-to-model transformation. Transformation synthesis is the common pattern for HOTs that generates transformations from different information sources [156]. These HOTs are defined by two conditions: 1) the output model is a transformation and 2) the input model is not a transformation.

Our HOT is also an exogenous transformation because its input and output models are expressed using different languages (decision and transformation models, respectively). Figure 5.2 shows the structure of HOT for generating tailoring transformations. The HOT is implemented in ATL and it is called VDM2ATL, since this transformation takes a variation decision model (VDM) as input and generates a transformation model as output. The VDM, that conforms to the variation decision metamodel (VDMM), is a formal specification of the tailoring rules. The transformation model that conforms to ATL metamodel is a formal specification of a transformation.

VDM2ATL is a transformation that uses ATL constructs for generating a transformation model. As mentioned before, the transformation model is a *module* that is divided into a *header* and a *body* section. VDM2ATL implements the header section as a set of matched rules, and the body section as a set of matched and called rules.

The header section declares the name and the source and target models of the transformation model. Listing 5.1 shows matched rules for defining the header section of the transformation model. This part of the HOT implements a *module* as ATLMeta!Module (line 8 in listing 5.1). ATLMeta!Module is an ATL construct that has attributes (that allow declaring a transformation name) as well as its source and target models. The transfor-

mation name is "TailoringTransformation" (name <- 'TailoringTransformation'), the input models are a software process model (inModels <- inputModels_spem) and a context model (inModels <- inputModels_scpm), and the output model is adapted software process model (outModels <- outputModels_spem). The transformation needs other ATL constructs, as Object Constraint Language (OCL) [1] for implementing expressions. For example inModels (line 15 in listing 5.1) requires ATLMeta!OclModel for defining a model's name and model's metamodel. Finally, the header section declares the rest of the rules that will be executed for generating the rest of the transformation model. The complete header section of the VDM2ATL is presented in Annex A, Section A.1 (see Listing A.1).

Listing 5.1: Excerpt of the HOT for Generating the Header Section

```
1  module VDM2ATL;
2  create OUT : ATLMeta from IN : VDMM;
3
4  rule Module {
5  from
6  vdm : VDMM!DecisionModel
7  to
8  atl : ATLMeta!Module (
9  isRefining <- false ,
10 name <- 'TailoringTransformation',
11 outModels <- outputModels_spem,
12 inModels <- inputModels_spem,
13 inModels <- inputModels_spcm
14 ),
15 outputModels_spem:ATLMeta!OclModel(
16 name <- 'OUT',
17 metamodel<- outputMetamodel_spem
18 ),
19 outputMetamodel_spem:ATLMeta!OclModel(
20 name <- 'MM2'
21 ),
22 inputModels_spem: ATLMeta!OclModel(
23 name <- 'IN',
24 metamodel <-inputMetamodel_spem
25 ),
26 inputMetamodel_spem:ATLMeta!OclModel(
27 name <- 'MM'
28 ),
29 inputModels_spcm: ATLMeta!OclModel(
30 name <- 'IN1',
31 metamodel <-inputMetamodel_spcm
32 ),
33 inputMetamodel_spcm:ATLMeta!OclModel(
34 name <- 'MM1'
35 )
36 ...
37 }
```

Figure 5.3 shows the header section that is generated from Listing 5.1. The header section is part of the transformation model that conforms to ATL metamodel and it can be visualized in Eclipse Modeling Framework (EMF). The header section declares the source and target models, which are typed by their metamodels. In our case, the source models are the organizational software process model (IN) that conforms to eSPEM (MM), and the organizational context model (IN1) that conforms to SPCM (MM1). The target model is the adapted software process model (OUT) that conforms to eSPEM (MM2).

The *body section* declares rules and helpers of the transformation model. Rules are matched rules for describing the transformation from a source model to a target model by relating metamodels. Helpers can be used to define (global) variables and functions, and they are OCL expressions. Helpers can call each other (recursion is possible) or they can be called from within rules.

Listing 5.2 shows matched rules for defining rules of the transformation model. This part of the HOT implements *matched rules* as ATLMeta!MatchedRule (line 4 in listing 5.2).

[1] OCL website: http://www.omg.org/spec/OCL/

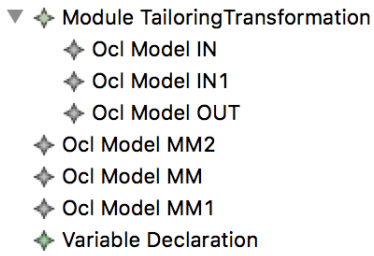


Figure 5.3: Header of the transformation model generated from the VDM2ATL.

ATLMeta!MatchedRule is an ATL construct that also allows declaring attribute rules, source pattern and target pattern. A source pattern consists of the keyword *inPattern* that defines a variable declaration and optionally a filter. A filter is an OCL expression restricting the rule to elements of the source model that satisfy certain constraints. A target pattern consists of the keyword *outPattern* that declares to which element(s) of the target model the source pattern has to be transformed. A target pattern element consists of a variable declaration (or more precisely the declaration of the target pattern variable) and a sequence of bindings (assignments). These patterns are defined as ATLMeta!InPattern (line 13 in listing 5.2) and ATLMeta!OutPattern (line 25 in listing 5.2), respectively.

ATLMeta!InPattern and ATLMeta!OutPattern require other ATL constructs for defining their elements as varName, type and bindings. The varName is "mpp" (varName <- 'mpp'), the type is an OCL expression (type <- outOclType) and the binding is an expression in terms of source or target binding (bindings <- bindingMethodPluginOutElement_name). The complete matched rules section of the VDM2ATL is presented in Annex A, Section A.2 (see Listing A.2).

Listing 5.2: Excerpt of the HOT for Generating Matched Rules

```
1  rule createMethodPlugin(vdm : VDM!DecisionModel){
2  to
3  -- InPattern y outPattern
4  methodPluginRule : ATLMeta!MatchedRule(
5  name <- 'methodPlugin',
6  isAbstract <- false ,
7  isRefining <- false ,
8  module <- thisModule.resolveTemp(vdm, 'atl'),
9  inPattern <- methodPluginInPattern,
10 outPattern <- methodPluginOutPattern
11 ),
12 -- From Section
13 methodPluginInPattern : ATLMeta!InPattern(
14 elements <- Set{methodPluginInElement}
15 ),
16 methodPluginInElement : ATLMeta!SimpleInPatternElement(
17 varName <- 'mp',
18 type <- inOclType
19 ),
20 inOclType : ATLMeta!OclModelElement(
21 name <- 'MethodPlugin',
22 model <- thisModule.resolveTemp(vdm, 'inputMetamodel_spem')
23 ),
24 -- To Section
25 methodPluginOutPattern : ATLMeta!OutPattern(
26 elements <- Set{methodPluginOutElement}
27 ),
28 methodPluginOutElement : ATLMeta!SimpleOutPatternElement(
29 varName <- 'mpp',
30 type <- outOclType,
31 bindings <- bindingsMethodPluginOutElement_name,
32 bindings <- bindingsMethodPluginOutElemen_description,
33 bindings <- bindingsMethodPluginOutElemen_ownedProcessPackage,
34 bindings <- bindingsMethodPluginOutElemen_ownedMethodContentPackage
35 ),
36 outOclType : ATLMeta!OclModelElement(
37 name <- 'MethodPlugin',
38 model <- thisModule.resolveTemp(vdm, 'outputMetamodel_spem')
39 ),
40 ...
```

Figure 5.4 shows matched rules section that is generated from Listing 5.2. The matched rules section is part of the transformation model that conforms to ATL metamodel, and it can be visualized in EMF. The matched rules section declares rules that establish the matching between the source and target models. In our case, we need matching rules for mapping the organizational software process model and the adapted software process model elements; for example: rule main, rule methodplugin and rule methodconfiguration.

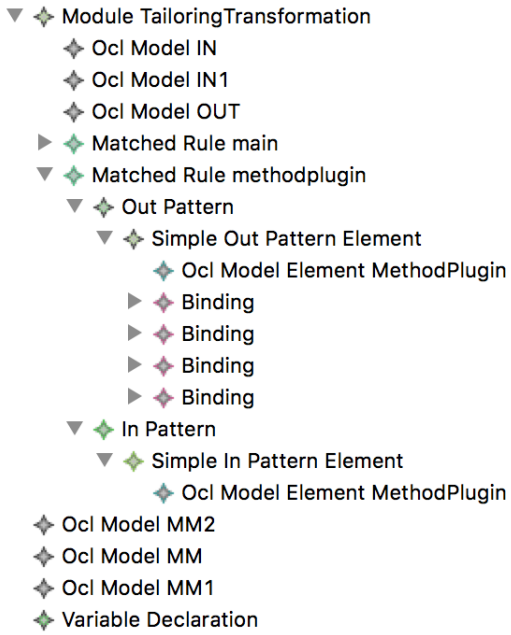


Figure 5.4: Mached rules of the transformation model generated from the VDM2ATL.

Listing 5.3 shows matched and called rules for defining helpers of the transformation model. This part of the HOT implements *called rules* as ATLMeta!Helper (line 3 in listing 5.3). ATLMeta!Helper is an ATL construct that allows declaring helpers and this definition. ATLMeta!Helper is defined in terms of other ATL constructors as operations, parameters, model elements, expressions and others attributes. ATLMeta!Helper has two attributes: module and definition. Module is an expression for referencing an ATL element that is defined by another rule. Definition is a helper's feature definition in terms of an OCL expression. These attributes are defined as keywords resolveTemp (line 4 in listing 5.3) and ATLMeta!OclFeatureDefinition (line 7 in listing 5.3), respectively.

ATLMeta!Operation is the helper definition that requires other ATL constructs for defining its elements: name, parameters, return type and body. The name element is a string that is "getContextAttributeConfiguration" (name <- 'getContextAttributeConfiguration'), the parameter element is an input operation in terms of an ATL type (parameter <- parameterOperation), the return type element is a return operation in terms of an ATL type or an OCL expression (returnType <- returnTypeOperation) and the body element is an expression in terms of another operation that includes iterations, variables and other expressions (body <- bodyOperation). These helper's elements are defined as ATLMeta!Operation (line 12 - 17 in listing 5.3), ATLMeta!Parameter (line 19 - 22 in listing 5.3), ATLMeta!OclModelElements (line 24 - 27 in listing 5.3) and ATLMeta!CollectionOperationCallExp (line 32 - 35 in listing 5.3), respectively. The complete matched and called rules sections for defining helpers of

the VDM2ATL is presented in Annex A, Section A.3 (see Listing A.3).

Listing 5.3: Excerpt of the HOT for Generating Helpers

```
rule helpergetContextAttributeConfiguration(vdm : VDMM!DecisionModel){
to
getContextAttributeHelper: ATLMeta!Helper(
module <- thisModule.resolveTemp(vdm, 'atl'),
definition<-definitionHelper
),
definitionHelper:ATLMeta!OclFeatureDefinition(
feature <- featureOclFeatureDefinition

),

featureOclFeatureDefinition: ATLMeta!Operation(
name <- 'getContextAttributeConfiguration',
parameters<-parameterOperation,
returnType<-returnTypeOperation,
body<-bodyOperation
),

parameterOperation: ATLMeta!Parameter(
varName <- 'nameAttribute',
type<-typeParameter
),

returnTypeOperation:ATLMeta!OclModelElement(
name<-'ContextAttributeConfiguration',
model<-thisModule.resolveTemp(vdm, 'inputMetamodel_spcm')
),

typeParameter:ATLMeta!StringType(
),

bodyOperation:ATLMeta!CollectionOperationCallExp(
operationName <-'first',
source<-sourceCollectionOperation
),
...
```

Figure 5.5 shows the section of matched and called rules generated from Listing 5.3. The called rules section is part of the transformation model that conforms to ATL metamodel and it can be visualized in EMF. The called rules section declares rules as called helpers that describe how to map model elements between the organizational software process model and the adapted software process model. In our case, we need called rules for describing the transformation rules that remove or replace software process elements, for instance the following helpers: getContextAttributeConfiguration, getTaskDefinition, optional rules and alternative rules.

The HOT is a generic transformation that generates a transformation model. To apply our HOT, we need to define a VDM and execute the HOT in Eclipse Modeling Tools (EMT) as an ATL project. Figure 5.6 shows an excerpt of a transformation model that can be visualized in EMF. The transformation element is automatically generated from the HOT.

The transformation model specifies elements as: ATL modules, matched rules, attributes, transformation rules and variable declarations. Each element requires a kind of constructor for defining its structure. For instance, *Matched Rule methodPlugin* is a transformation element that requires two constructors:*Out Pattern* and *In Pattern*. Figure 5.6a highlights *Matched Rule methodPlugin* that is defined from our HOT. Figure 5.6b highlights *Out Pattern* that is a constructor of *Matched Rule methodPlugin*. In both figures we can see a set of properties that are generated from our HOT.

Finally, the HOT can be used for generating any transformation model from a particular VDM.

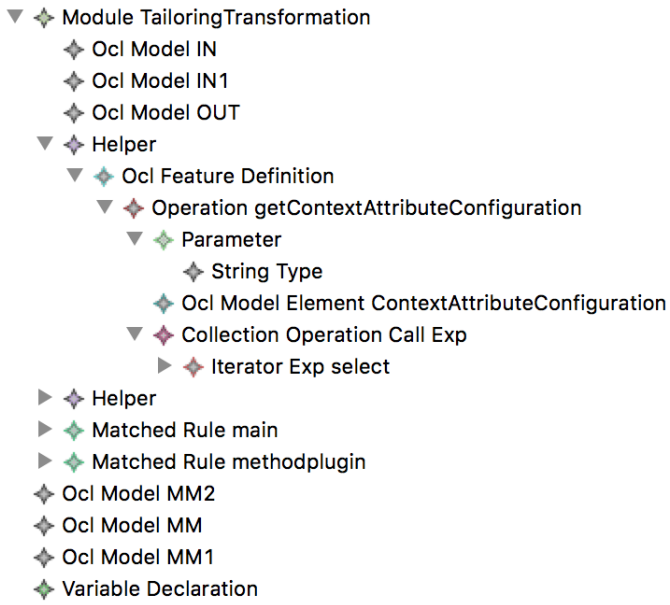


Figure 5.5: Matched and called rules of the transformation model generated from the VDM2ATL.

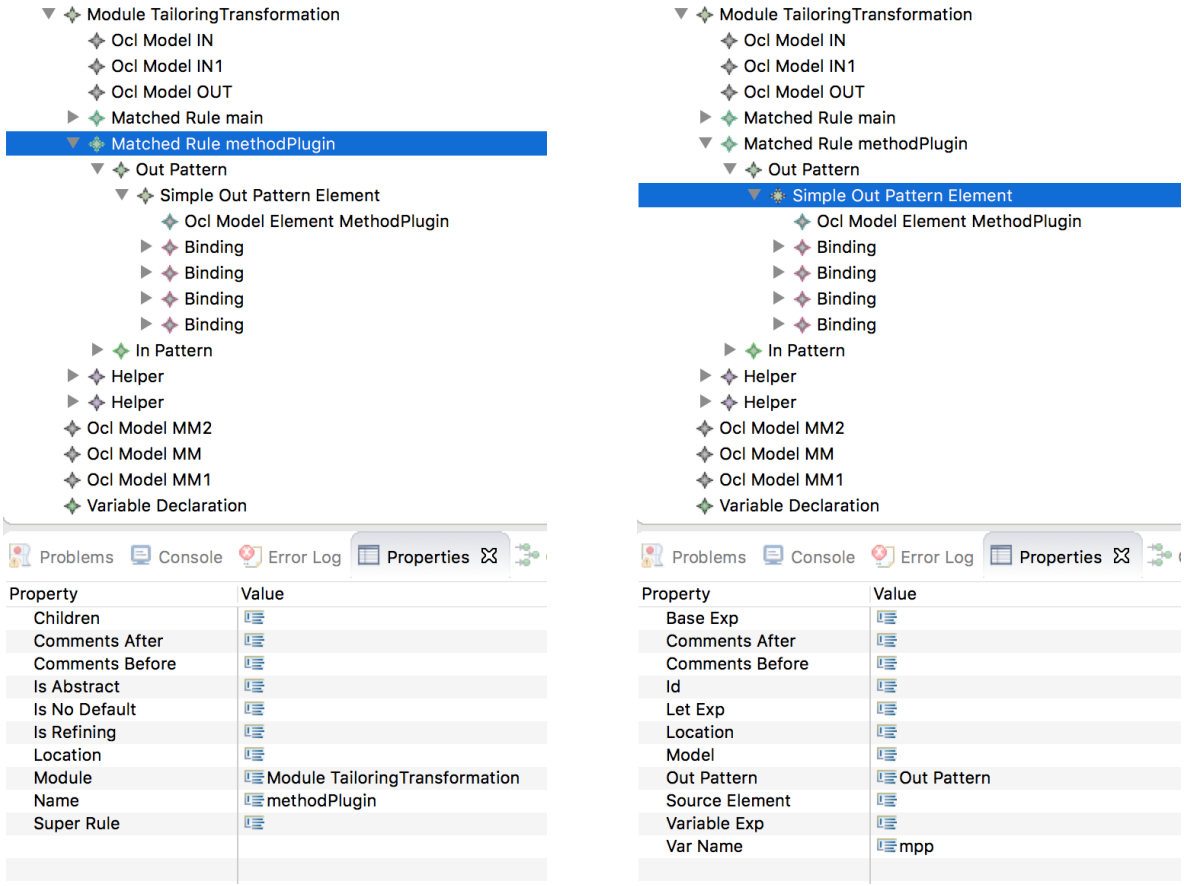
5.3.2 Benefits and drawbacks of the HOT

Generating the transformation model automatically based on a variation decision model and a higher-order transformation instead of direct manipulation of code separates the transformation generation from tailoring rules specification. This separation can achieve high variability and flexibility in the generation of transformation models because the HOT is generic to any VDM. The HOT does not get polluted with ATL code that is only responsible for checking the input model. Moreover, this alleviates the complexity of transformation evolution.

The main benefit of the proposed HOT for this scenario is the generation of transformation models. In this sense, the transformation model is generated in an automatic manner, and it is well-formed according to the evaluation results. Additionally, the generated transformation models are better structured and therefore more understandable.

According to Syriani [152], the verification of well-formed higher-order transformation is quite limited. In this case, with a correctness and completeness mechanism at the end of the transformation level, the developer of HOT may test that the target model produced by the transformation is well-formed in a set of test cases. In our application domain (for software process tailoring), we can compare the transformation model expected and the transformation model generated. Moreover, the Eclipse Modeling Tool (EMF) considers a set of tools for verifying the syntactic correctness when a transformation model is generated. So far, we have not generated incorrect transformations in practice.

Despite the advantages in simplifying the generation of transformation models with our HOT, there are also some drawbacks that need to be discussed. First, the execution of our HOT should be executed in EMF as ATL project. Second, the potential users need to configure the ATL project for generating a new transformation model using other VDM.



(a) Elements and properties of the *Matched Rule methodPlugin*.

(b) Elements and properties of the *Out pattern* that is part of the *Matched Rule methodPlugin*.

Figure 5.6: Excerpt of a transformation model generated from our HOT.

Third, the transformation model is saved as *ecore* in a file format for ATL model encoding.

In order to address the drawbacks, we propose a graphical environment for executing and configuring the HOT, and an ATL extractor for generating ATL code from an *ecore* file. The graphical environment will be presented in section 5.6 and the ATL extractor will be presented in the next section 5.4.

5.4 ATL Extractor for Generating Transformation Code

In the MDE-based approach, the transformation is implemented in ATL code and it includes tailoring transformation rules for software process tailoring.

After applying our HOT, we can obtain a transformation model that includes tailoring rules, and it can be used for generating ATL code using an extractor. Figure 5.7 shows an extractor that takes a transformation model as input and generates ATL code as output. This code generation is a vertical transformation because its source and target models are at different levels of abstraction (model representation and code generation). The extractor is

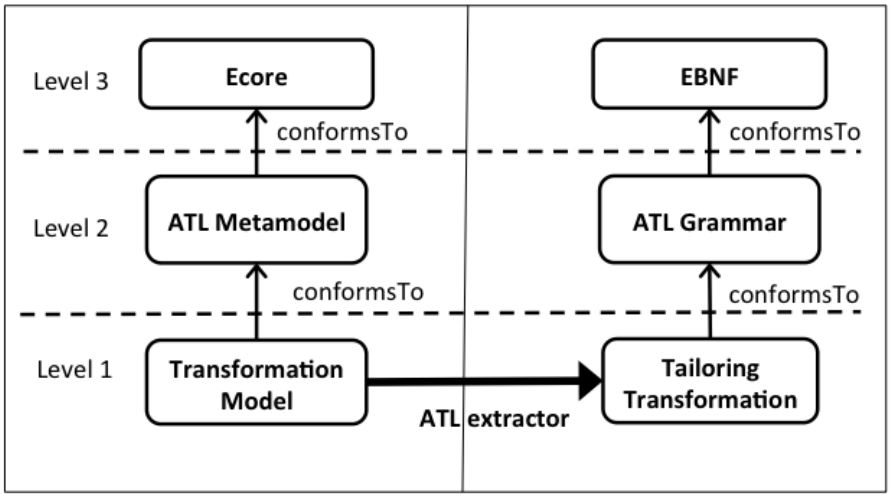


Figure 5.7: Extractor for generating tailoring transformation as ATL code.

called *ATL extractor*.

The ATL extractor is a plugin developed by AtlanMod MegaModel Management (AM3) project [2]. This extractor takes an ATL model as input and automatically generates ATL code. The ATL model is generated by our HOT and the ATL code is the tailoring transformation. Moreover, the ATL extractor can also be executed in EMT.

The tailoring transformation is generated only once, and it can be used for MDE-based tailoring approach with different project contexts generating a particular software process for each of them. Listing 5.4 shows an excerpt of ATL code generated by the ATL extractor.

Listing 5.4: Excerpt of the ATL code part of the tailoring transformation

```
1  module TailoringTransformation;
2  create OUT : MM2 from IN : MM, IN1 : MM1;
3
4  helper def: getContextAttributeConfiguration(nameAttribute : String) : MM1!ContextAttributeConfiguration =
5  MM1!ContextAttributeConfiguration.allInstances()->as Sequence ()->select(a |
6  a.myContextElement.name = nameAttribute)->first();
7
8  helper def: optionalRule(name : String) : Boolean =
9  if (Sequence {'Requirements'}.includes(name)) then
10 (if ('Requirements'=name) then
11 thisModule.ruleOpt1())
12 else
13 true
14 endif;
15
16 helper def:ruleOpt1(): Boolean =if(thisModule.getValue('Project Type') = 'Maintenance-adaptation' and
       thisModule.getValue('Project Duration') = 'Small') then false
17 else true
18 endif;
19
20 rule main {
21 from
22 ml : MM!MethodLibrary
23 to
24 mll : MM2!MethodLibrary (
25 name <- ml.name,
26 description <- ml.description,
27 ownedMethodPlugin <- ml.ownedMethodPlugin,
28 predefinedConfiguration <- ml.predefinedConfiguration
29 )
30 }
31
32 rule methodPlugin {
33 from
34 mp : MM!MethodPlugin
35 to
```

[2] AM3 website: https://wiki.eclipse.org/AM3

```
36 mpp : MM2!MethodPlugin (
37 name <- mp.name,
38 description <- mp.description,
39 ownedProcessPackage <- mp.ownedProcessPackage,
40 ownedMethodContentPackage <- mp.ownedMethodContentPackage
41 )
42 }
43 ...
44 ...
45 ...
```

5.5 Using the Tailoring Transformation Generation

In Chapter 3, we defined a running example to illustrate the MDE-based tailoring. The running example considered part of Rhiscom's software process and showed a tailoring transformation in ATL code. The tailoring transformation implemented two decisions for removing two optional process elements: *Design* and *Requirements*; and other two decisions for replacing two alternative process elements: *Specify requirements* and *Establish requirements baseline*.

We use our HOT and ATL extractor for automatically generating this tailoring transformation. In this sense, we need to apply two steps: (1) generate a transformation model using the HOT, (2) generate a tailoring transformation using the ATL extractor.

5.5.1 Generating a transformation model using the HOT

The HOT takes the VDM as input for generating the tailoring transformation model. To that end, the user needs to configure and execute the HOT in EMF. An ATL launch configuration aims to resume all the information that is required to execute an ATL transformation. Figure 5.8 shows an ATL launch configuration for executing our HOT. This mainly includes the path of the files involved in the transformation (the transformation file, the input and output models, the input and output metamodels, the libraries). In this sense, the ATL launch configuration is the following:

- *ATL Module* section enables specifying the transformation file. The transformation file is *VDM2ATL.atl*
- *Metamodels* section enables specifying the source and target metamodels. The source metamodel is the *VDMM* and its name is *variationDecisionMetamodel.ecore*. The target metamodel is the *ATLMeta* and its name is *atl.ecore*.
- *Source Models* section enables specifying the source model that conforms to source metamodel. The source model is the VDM that conforms to VDMM and its name is *variationDecisionModelRhiscom.xmi*
- *Target model* section enables specifying the target model that conforms to the target metamodel. The target model is the ATL model that conforms to ATLMeta and its name is *atlTransformationModelRhiscom.ecore*

The *ATL Module* and *Metamodels* have the same name and path in any case. However,

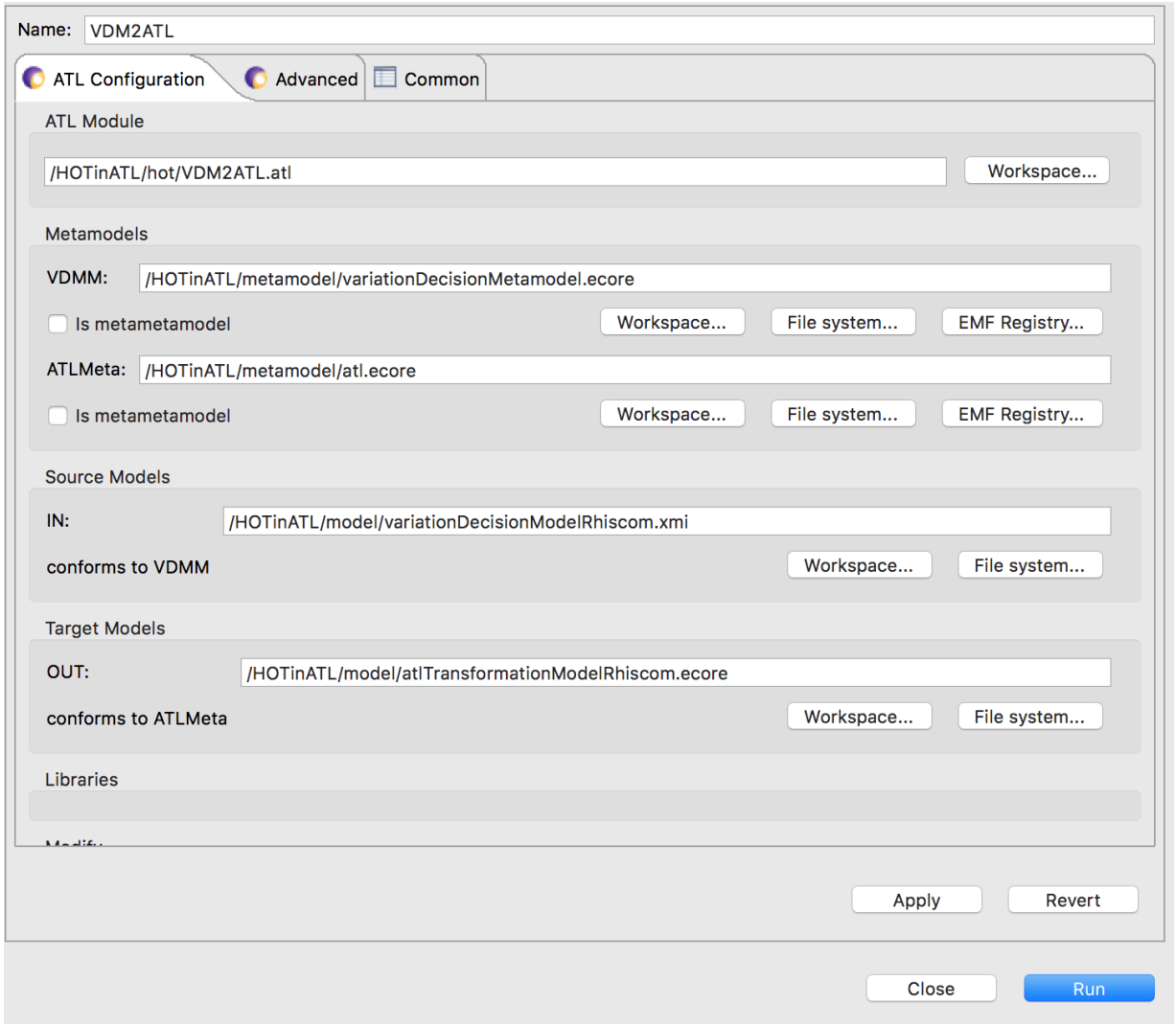


Figure 5.8: ATL configuration in EMF for executing the HOT.

the *Source Model* can have another name and path because it was generated by the graphical environment for defining tailoring rules described in Chapter 4. On the other hand, the *Target model* can have another path and name but it should have *ecore* extension that is compatible with the ATL extractor.

Finally, the *Run* button executes the HOT and generates the tailoring transformation model. Figure 5.9 shows the ATL model that is the tailoring transformation model of Rhiscom software company described in Chapter 3.

5.5.2 Generating a tailoring transformation using the ATL code extractor

The ATL extractor takes the tailoring transformation model and generates tailoring transformation code. The ATL extractor is a plugin that can be executed in EMF. To that end,

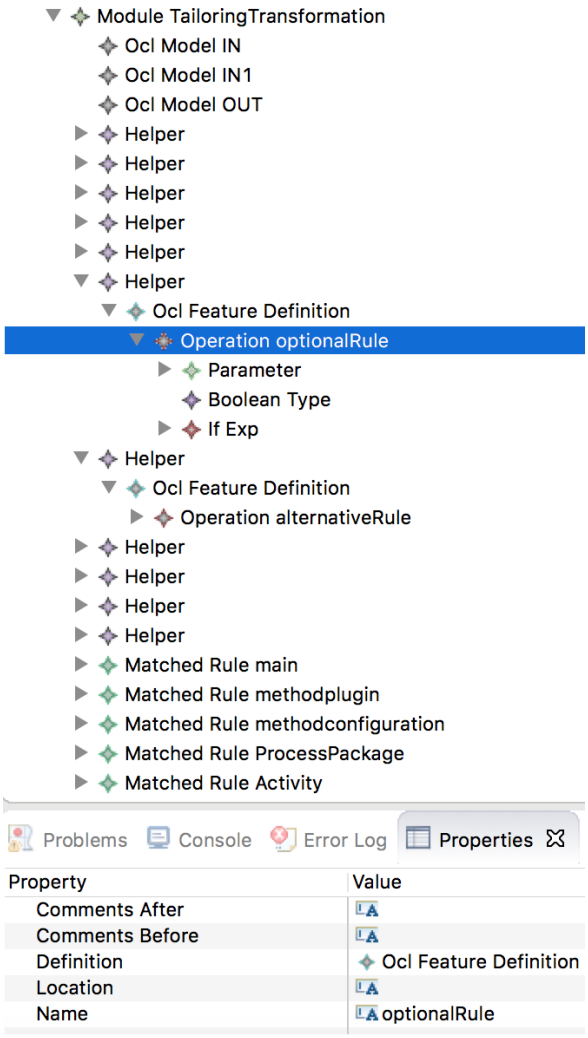


Figure 5.9: Tailoring transformation model of Rhiscom.

the user needs to follow the following actions: (1) Select the transformation model in *Project Explorer*, (2) Execute the pop up menu of transformation model, and (3) Select the Extract ATL code from Model. Figure 5.10 shows the execution of the ATL extractor in EMF for extracting the transformation code from the tailoring transformation model of Rhiscom. To that end, the user selects *atlTransformationModelRhiscom.ecore* that is the tailoring transformation model and executes *Extract ATL-0.2 model to ATL-0.2 file* that generates the transformation as ATL code. Finally, the ATL extractor automatically generates *atlTransformationModelRhiscom.atl* that can be used for applying the MDE-based software process tailoring.

Listing 5.5 shows an excerpt of the tailoring transformation that is generated using the HOT and the ATL extractor. The complete tailoring transformation as ATL code is presented in Annex A, Section A.4 (see Listing A.4).

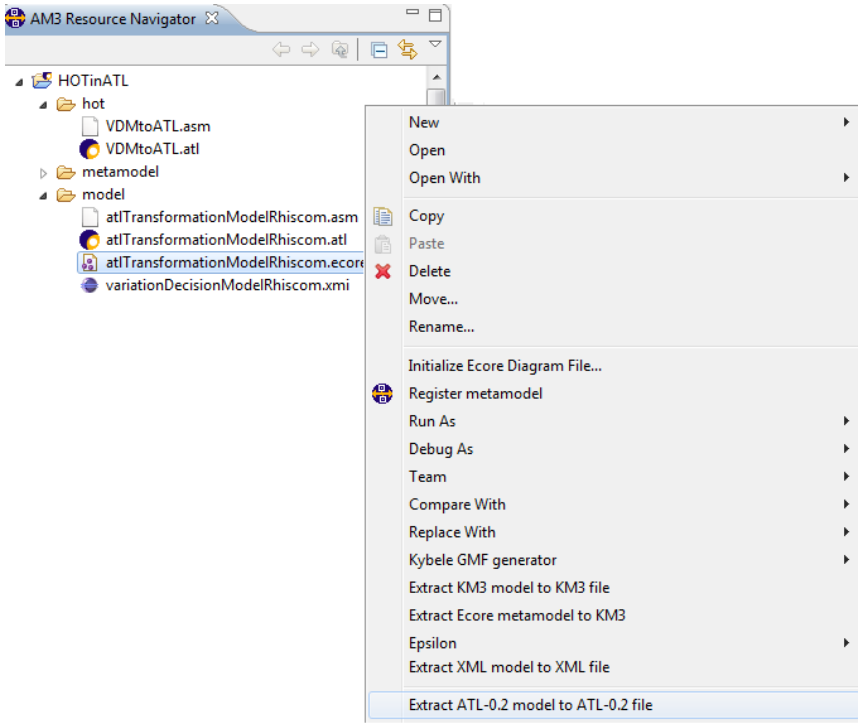


Figure 5.10: Extracting the transformation code of Rhiscom.

Listing 5.5: Tailoring transformation generated by the HOT and the ATL extractor

```
1  module Tailoring;
2  create OUT : MM2 from IN : MM, IN1 : MM1;
3
4  helper def: getContextAttributeConfiguration (nameAttribute: String ) :
5  MM1!ContextAttributeConfiguration = MM1!ContextAttributeConfiguration.allInstances()->as Sequence ()->select(
       a | a.myContextElement.name =nameAttribute )->first();
6
7  helper def: getValue(nameAttribute: String ):
8   String = thisModule.getContextAttributeConfiguration(nameAttribute).myContextAttributeValue.value;
9
10 helper def:getTaskDefinition(taskDefinitionName: String ): MM!TaskDefinition =
11 MM!TaskDefinition.allInstances()->as Sequence ()->select(t|t.name = taskDefinitionName)->first();
12
13 helper def:nextElement(a:MM!WorkBreakDownElement): MM!WorkBreakDownElement = MM!WorkBreakDownElement.
       allInstances()->select(t|t=a.next)->first();
14 helper def:next(a:MM!WorkBreakDownElement): MM!WorkBreakDownElement = if(thisModule.optionalRule(thisModule
       .nextElement(a).name)) then a else thisModule.next(thisModule.nextElement(a)) endif;
15
16 helper def: optionalRule(name: String ): Boolean =
17 if( Sequence {'Requirements','Design') then(
18 if('Design'= name) then
19 thisModule.ruleOpt2()
20 else (
21 if('Requirements'= name) then
22 thisModule.ruleOpt1()
23 else true
24 endif
25 ) endif
26 ) else true endif;
27
28 helper def: alternativeRule(tu:MM!TaskUse): MM!TaskDefinition =
29 if( Sequence {'Specify Requirements', 'Establish Requirements Baseline'}.includes(tu.name)) then(
30 if('Establish Requirements Baseline'= tu.name) then
31 thisModule.ruleAlt2(tu)
32 else (
33 if('Specify Requirements'= tu.name) then
34 thisModule.ruleAlt1(tu)
35 else tu.linkTask
36 endif
37 ) endif
38 ) else tu.linkTask endif;
39
40 helper def:ruleOpt1(): Boolean =if(thisModule.getValue('Project Type') = 'Maintenance-adaptation' and
       thisModule.getValue('Project Duration') = 'Small') then false
41 else true
42 endif;
43 ...
44
45 rule main{
```

```
46 from      ml:MM!MethodLibrary
47 to                mll:MM2!MethodLibrary(
48 name <- ml.name,
49 description <- ml.description,
50 ownedMethodPlugin<- ml.ownedMethodPlugin,
51 predefinedConfiguration<-ml.predefinedConfiguration
52 )
53 }
54 rule methodplugin{
55 from      mp:MM!MethodPlugin
56 to                mpp:MM2!MethodPlugin (
57 name<- mp.name,
58 description <- mp.description,
59 ownedProcessPackage<- mp.ownedProcessPackage,
60 ownedMethodContentPackage <- mp.ownedMethodContentPackage
61 )
62 }
63 ...
```

5.6 Graphical Environment for Generating Tailoring Transformations

In order to improve the usability of the HOT and ATL extractor, we developed a graphical environment for generating the tailoring transformation. The process engineer uses the graphical environment for selecting the VDM and automatically generating the ATL configuration. After that, she/he defines a name for the tailoring transformation and automatically executes the HOT for generating the tailoring transformation as ATL code.

To that end, we developed an *ATL Launcher* for configuring and executing ATL transformations. The ATL Launcher uses Java libraries of ATL and Eclipse Modeling Framework (EMF) that are available in EMT. EMF is a modeling framework and code generation facility for building tools and other applications based on a structured data model. From a model specification described in XMI, EMF provides tools and runtime support to produce a set of Java classes for the model, along with a set of adapter classes that enable viewing and command-based editing of the model, and a basic editor.

The ATL Launcher can be integrated with a standalone Java project that can be reused to run ATL model-to-model transformations programmatically. In this sense, we can configure and execute the HOT and ATL extractor without interacting the EMT.

Figure 5.11 shows a graphical environment for generating tailoring transformations. The graphical environment includes the HOT and ATL extractor that are executed using the ATL Launcher. In this sense, the user only needs to select the variation decision model and the graphical environment generates the tailoring transformation as ATL code.

Finally, the tailoring transformation is saved in the same workspace of the VDM. The tailoring transformation can be used for software process tailoring according to Hurtado proposal [65].

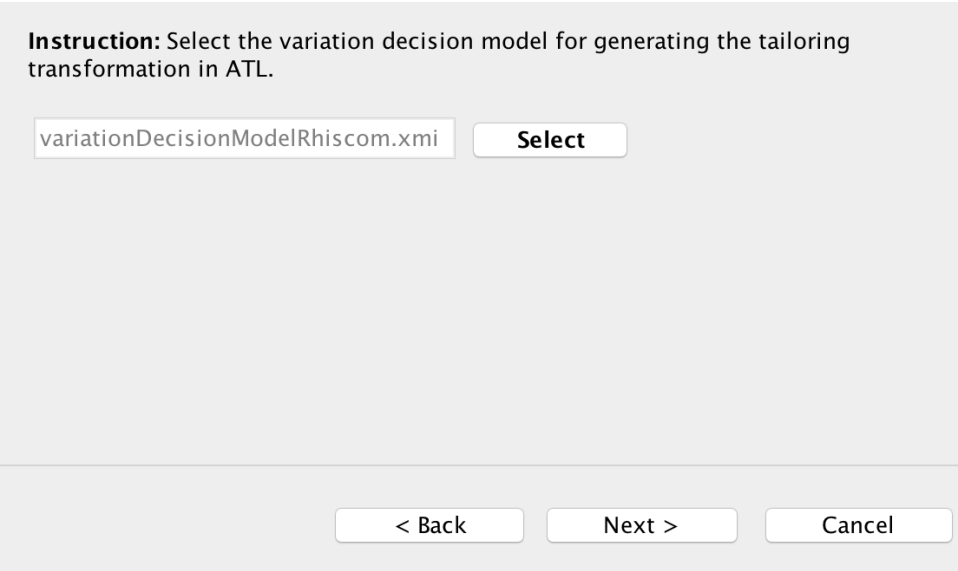


Figure 5.11: Graphical Environment for Generating Tailoring Transformations.

5.7 Related Work

HOTs are transformations that may produce other transformations as output. This is an advantage, as instead of writing transformations manually, one can write a transformation that automatically generates the desired transformation. There is still no standard language for developing HOTs, and there are not many successful real world implementation experiences reported either.

The most widely used language to build HOTs is ATL. The literature also reports that there are other languages that allow implementing HOTs, such as GReAT [4], MOFScript [108] and XSLT [26]. Tisi et al. [156] use ATL to define patterns for building HOTs, and establish transformations aimed at improving their use. Also, they present a summary of HOTs for different applications.

According to Tisi et al. [156], the knowledge for generating HOTs is verbose, i.e., it requires a long and complex code for generating just a small output transformation. This is mainly the problem with technological approaches for building HOTs, because they are always complex. In this sense, the HOTs have a generic structure that can be reused in other domains. However, there are not HOTs for generating tailoring transformations in the software process domain.

The literature reports a few HOTs for implementing model transformations. Tisi et al. [156] presented a categorized survey of HOTs. The categorization was divided based on some variant types of the four base transformation patterns (synthesis, analysis, composition and modification). Table 5.1 shows a list of published HOTs considering source and target metamodels and transformations types, and it highlights other HOTs that were found in the literature. The list of HOTs is adapted from Tisi et al. [156] and considered the same variant types that were defined in the original characterization.

Table 5.1: Summary of HOTs (adapted and updated from Tisi et al. [156]).

Name (# of cases)	Language	Source MM	Target MM	Type	Ref.	Year
AMWtoATL_KM32SQL	ATL	AMW	ATL	Implementation	[124]	2005
AMWtoATL_MantisBug	ATL	AMW	ATL	Implementation	[35]	2006
AMWtoATL	ATL	AMW	ATL	Implementation	[123]	2006
AMWtoXSLT	ATL	AMW	XSLT	Implementation	[123]	2006
ATL2BindingDebugger	ATL	ATL	ATL	Weaving	[119]	2005
ATL2Tracer	ATL	ATL	ATL	Weaving	[74]	2005
ATL2WTracer	ATL	ATL	ATL	Weaving	[118]	2006
ATL2Problem	ATL	ATL	Problem	Analysis	[121]	2005
KM32CONFATL	ATL	KM3	ATL	Generic	[51]	2007
KM32ATLCopier	ATL	KM3	ATL	Generic	[122]	2005
AMWtoATL_Kelly	ATL	AWM	ATL	Implementation	[47]	2005
MMD2ATL	ATL	KM3	ATL	Implementation	[25]	2007
MMTtoMT	ATL	ATL, Ecore	ATL	Execution	[20]	2008
MSDSL2EMF	ATL	KM3	ATL	Generic	[13]	2005
Superimpose	ATL	ATL, ATL	ATL	Composition	[166]	2008
ATLCopy	ATL	ATL, ATL	ATL	Composition	[166]	2008
HITransform	MOFScript	MOFScript	MOFScript	Variants	[107]	2007
Topcased	ATL	ATL	ATL	Analysis	[39]	2006
Metamodel2Derivation	ATL	Ecore	ATL	Generic	[16]	2007
DUALLyLeft2Right (2)	ATL	AMW, Ecore, Ecore	ATL	Implementation	[97]	2010
Easystyle	XSLT	HTML	XSLT	Implementation	[168]	2008
HOT4Tests	ATL	Ecore	ATL	Testing	[21, 19]	2008
Mutators (11)	ATL	ATL, Trace	ATL	Mutation	[21, 19]	2008
SingleApplication	ATL	ATL	ATL	Execution	[21, 19]	2008
patchgen	ATL	AMW	ATL	Implementation	[34]	2007
propagate	ATL	ATL, INMM, INMM, AMW	ATL	Implementation	[34]	2007
VariabilityMM_HOT	GREaT	GME, GReAT	GReAT	Analysis	[80]	2008
MML2MMR (2)	ATL	AMW, Ecore, Ecore	ATL	Implementation	[72]	2007
AML2ATL	ATL	AML	ATL	Extension	[48]	2009
UITransReconfig	ATL	ATL, USRMM	ATL	Adaptation	[149]	2007
Ecore2RDF (2)	ATL	Meo, Ecore, OWL	ATL	Implementation	[59]	2008
ATL2BindingDebugger	ATL	ATL, Ecore	ATL	Testing	[120]	2007
QVT2ATL	ATL	QVT	ATL	Generic	[73]	2007
CDM2ATL	ATL	CDM	ATL	Implementation	[172]	2012
MOFScript2MOFScript	MOFScript	MOFScript, HandyMOF	MOFScript	Testing	[49]	2014
Metamodel2Derivation	ATL	DSL_MM	ATL	Implementation	[17]	2009
MTL2T-Core	Motif	MTL	T-Core	Implementation	[37]	2014
DSL_MM2ATL	MOFScript	DSL_Transfo	ATL	Implementation	[69, 50]	2010
DSML_MM2Motif	Motif	DSL_MM	DSL_MM	Generic	[160, 153]	2013

In Table 5.1, we can see that the most widely used language to build HOTs is ATL. Moreover, we can see a set of common transformation types that involve HOTs: implementation, weaving, analysis, generic, execution, composition, variants, testing, mutation and adaptation. In this sense, we found that the similar transformation types for building a *tailoring transformation* are implementation and weaving.

Implementation is the abstract representation that can be considered as the higher-level specification of a mapping that needs to be implemented as a transformation [156]. This type of transformation is used for enabling tool interoperability between different systems or languages. For example, Didonet del Fabro al. [35] report the use of ATL to create a transformation that uses a implementation mapping to transform a bug tracking model in Mantis, into an equivalent model in Bugzilla (bug-tracking systems). The authors also provide other examples using Atlas Model Weaver (AMW) [36] (mapping between the source and the target model) and ATL to generate a model transformation that translates a Kernel Metametamodel (KM3) into the Structured Query Language (SQL) data definition language, and vice versa. In both cases an AMW mapping that correlates two different metamodels is translated into two transformations at model level (one for each direction of the mapping).

Weaving is used to weave cross-cutting concerns into a model transformation. Examples of these concerns are related to debugging, traceability, program tracing [156]. A HOT for adding a cross-cutting concern can usually be programmed with extreme generality, and complete independence from the logic of the original transformation. In this way the same HOT can be used as a general means to add that specific feature to any transformation. There are several cases in literature that exploit ATL HOTs for cross-cutting concerns. ATL project [119] describes an ATL2BindingDebugger HOT that adds a debug instruction to each attribute binding in an input ATL transformation. Jouault [74] reports two different weaving implementations for another application case, in which the addressed cross-cutting concern is traceability, i.e., the maintenance of a set of links between the corresponding source and target model elements.

There are also proposals such as Model Transformations By Example [163] and Model Transformation By Demonstration [151], which present solutions for simplifying the implementation of model transformations by using strategies and patterns with a visual support. They both generate part of the transformation code, but the user still needs to complete the generated code, and thus the transformation rule generation becomes a semi-automatic process. Both proposals highlight the need for developing new solutions that simplify the rule definition, so that they become usable solutions for non-experts in model-driven engineering and model-driven development.

5.8 Summary and Discussion

The literature reports few numbers of HOTs for generating tailoring transformations using semi-automatic generators. A promising approach for developing HOTs is the Atlas Model Weaver (AMW) [36]; it defines a mapping model between the source and target model, and this mapping model is automatically translated into the model transformation using a HOT.

The purpose of AMW is to generate transformations for traceability or matching, so the transformation rules are simple and they do not include complex structures. In particular, it is not possible to include conditions for matching elements as we need for software process tailoring. Nevertheless, we follow the structure of the AMW for defining a relationship between the organizational software process model and the adapted software process model, but using the VDM that is a kind of mapping model with complex structures and constructors. The VDM can be used as an input model for a HOT and generating a tailoring transformation.

The proposed HOT takes a VDM that conforms to VDMM as input and generates a transformation model that conforms to the ATL metamodel, using a transformation synthesis pattern. This HOT is an exogenous transformation because the input and output models conform to different metamodels (decision and transformation models). Finally, we apply an ATL extractor for generating the tailoring transformation from the tailoring transformation model. The ATL extractor applied a vertical transformation because the source and target model reside at different abstraction levels, thus the tailoring transformation model is translated into the tailoring transformation source code.

Chapter 6

A Tool-set for Automatically Generating Tailoring Transformations - ATAGeTT

This chapter presents an integrated tool-set for automatically generating tailoring transformations and executing software process tailoring. The integrated tool set allows process engineers to define the organizational software process and the organizational context using the software context modeling tool, as well as tailoring rules using a graphical environment, so that the project manager just requires defining the project context of the particular project using the software context modeling tool in order to automatically obtain the tailored software process. This tool-set hides the complexity of MDE components and the users only require the knowledge about project characteristics and how they affect tailoring.

Section 6.1 introduces a motivation for specifying tailoring rules. Section 6.2 shows an overview of megamodeling. Section 6.3 presents the integrated tool for generating tailoring transformations and executing software process tailoring. Section 6.4 illustrates the integrated tool using an example.

6.1 Motivation

In this thesis, we present the tailoring rules specification (see Chapter 4) that allows process engineers to specify tailoring rules as a model using our decision language (DL), and the tailoring transformation generation (see Chapter 5) that allows users to generate transformations as an ATL code using our HOT and an ATL extractor. Although the tailoring rules specification and tailoring transformation generation are supported by tools, the users need to interact with models, transformation and configurations in EMF for executing software process tailoring. Moreover, the tools have several dependencies with other tools as the context modeling tool and projectors (see Chapter 2) that jeopardize its usage in software industry because the users need to use different tools.

To improve the integration of these tools, we propose A Tool-set for Automatically Generating Tailoring Transformations (ATAGeTT). It provides an integrated tool that allows

users -the process engineer and the project manager- to deal only with the software process and context-related concepts hiding and integrating all dependencies involved in software process tailoring [143, 144]. With the tool-set process engineers can define organizational contexts, project contexts and tailoring rules, and project managers can execute software process tailoring. All these activities are executed without the need of manipulating models or transformations.

6.2 An Overview of Megamodeling

Practical applications of MDE are increasingly intensive in modeling artifacts. They involve a large number of heterogeneous and interrelated models which change over time. Megamodeling is the model-based approach for coping with the complexity of managing and evolving such large model repositories [6]. It is centered on the notion of megamodel introduced by Bezivin et al. [14], which conveys the idea of modeling-in-the-large by establishing and using the global metadata and relationships on the modeling artifacts while ignoring their internal details. Global Model Management [100] is a megamodeling approach that offers Eclipse-based tool support for managing modeling artifacts. It has a metamodel that characterizes the different kinds of modeling artifacts and their interrelations.

6.3 Integrated Tool-set for Automatically Generating Tailoring Transformations and Executing Software Process Tailoring

The main stakeholders in software process tailoring are: 1) the process engineer, who is in charge of defining, evaluating and evolving the software process, and 2) the project manager, who follows the tailored software process during a project. We provide two integrated tools, one for each stakeholder, that support their activities and that hide the complexities of the megamodel. Figure 6.1 shows the structure of our complete model-based process tailoring solution using a megamodel as back end and user interfaces as front end. The megamodel includes models, their corresponding metamodels, and text-to-model (T2M), model-to-model (M2M), model-to-text (M2T) transformations, as well as higher-order transformations [146].

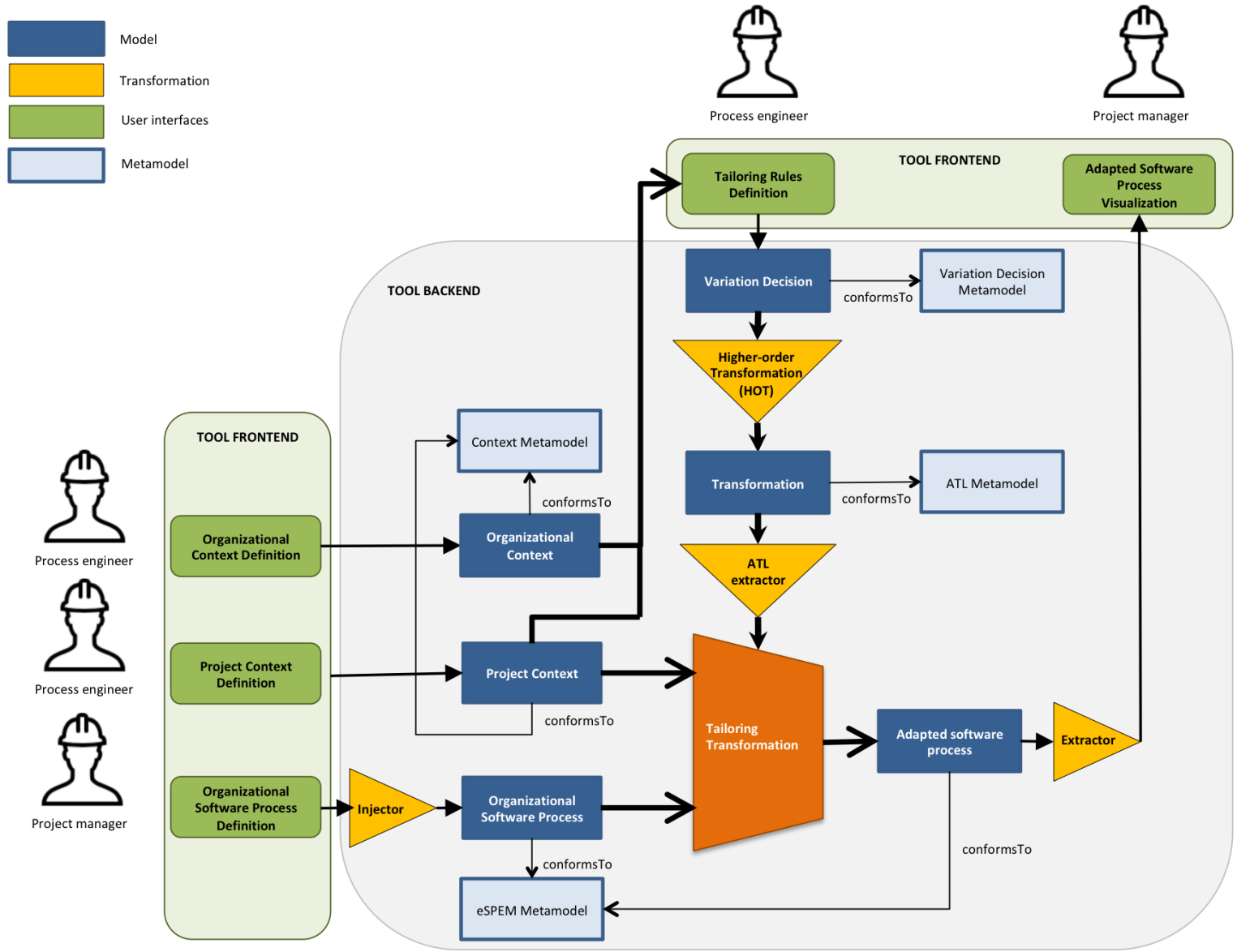


Figure 6.1: Structure of the megamodel along with the user interfaces

6.3.1 User interfaces

Process Engineer's Interface. The company's process engineer is in charge of the *Organizational Software Process Definition*. We use the Eclipse Process Framework (EPF) for this task, which allows the specification of software process models in SPEM 2.0 (Software and Systems Process Engineering Metamodel). The process engineer must also indicate the variation points of the software process model.

Project characteristics determine which is the most appropriate process for a project; these characteristics can be modeled as a project "context". We use the software context modeling tool (defined in Chapter 2) for the *Organizational Context Definition*; the process engineer uses this tool for defining the context attributes that affect the variable elements of a process, as well as their potential values, generating the organizational context model as output.

The most challenging task for the process engineer is the *Tailoring Rules Definition*, i.e., specifying the relationships between context attributes and variable process elements. These relationships define how variability is resolved during software process tailoring. In this definition, we use the graphical environment for defining tailoring rules and generating tailoring transformations (defined in Chapters 4 and 5), generating the variation decision model, the transformation model and the tailoring transformation as ATL code.

Project Manager's Interface. The project manager is in charge of the *Project Context Definition* using the software context modeling tool (defined in Chapter 2) that takes the organizational context model as input and allows her/him to choose the particular values for the attributes of the project at hand, generating the project context model as output. Moreover, the tool allows project managers to execute the software process tailoring using an adaptation of the ATL Launcher (defined in Chapter 5), generating the adapted software process model.

The EPF is used for *Adapted Software Process Visualization*, allowing the project manager to examine the tailored process to be followed during the project.

6.3.2 Megamodel

The elements in this section are explained in the order that they are usually defined or generated. The input models are modified directly by the user using tools, while rest of the models are generated from these input models using mapping transformations.

Organizational Software Process Model. The *Organizational Software Process Definition* can export a software process as XML, so we use the *Injector* (defined in Chapter 3) to generate the organizational *Software Process* model from this type of XML file. This model conforms to eSPEM (defined by Hurtado et al. [65]). In this metamodel, tasks, roles and work products represent optional process elements by setting the value of a Boolean attribute, and alternatives are specified by linking variation points to possible realizations using the *replaces* SPEM variability primitive.

Organizational Context Model. The *Organizational Context Definition* can generate an *Organizational Context* as XML. The Software Process Context Metamodel (SPCM) is a metamodel that has two sections: Context and Context Configuration (defined by Hurtado et al [65]). We use the Context section for specifying the *Organizational Context*. The organizational context model is a set of attributes and their potential values that can be used for characterizing project contexts. These attributes can be grouped in dimensions for better comprehension.

Variation Decision Model. The *Tailoring Rules Definition* establishes the relationship between context attribute values and variable process elements as *Variation Decision* model (defined in Chapter 4) using the Decision Language (DL). The variation decision model is a series of <condition, conclusion> pairs that conforms to Variation Decision Metamodel (VDMM).

Transformation Model. The *Tailoring Rules Definition* includes the generation of *Tailoring Transformation*. We use the *Higher-order Transformation (HOT)* (defined in Chapter 5) that automatically generates the *Transformation* model. This HOT takes the variation decision model as input and generates the *Transformation* model as output. The transformation model conforms to ATL metamodel.

Project Context Model. The *Project Context Definition* allows generate the *Project Context* model. We use the Context Configuration section of SPCM for specifying the *Project Context* model. The project context model is a valid configuration of the organizational context model, where each attribute has been assigned a value.

Tailoring Transformation. We use the *ATL extractor* (defined in Chapter 5) for generating the *Tailoring Transformation*. The *Tailoring Transformation* takes the *Organizational Software Process* model and the *Project Context* model as input, and produces a *Adapted-Software Process* model as output, which also conforms to eSPEM.

Adapted Software Process Model. The tailoring process generates the *Adapted Software Process* model, but this model cannot be directly manipulated by the project manager because it is not in a format supported by the tools that she/he uses. To remedy this we use the *Extractor* (defined in Chapter 3) to transforms the *Adapted Software Process* from its XMI format back into its original XML format, so that it can be imported by the EPF for *Adapted software Process Visualization*.

6.4 Building Adapted Software Process with ATAGeTT

ATAGeTT involves the front end and back end for a complete model-based process tailoring solution. Our integrated tool provides interfaces for process engineers and project managers. We show the application of the integrated tool using the running example defined in Chapter 2.

6.4.1 The process engineer user interface

The process engineer is in charge of defining the organizational process model, the organizational context model, and the variation decision model, i.e., the model that represents the tailoring rules. Our tool integrates user interfaces for performing each of these activities.

The process engineer is the one who designs/defines:

- The software process along with its variable elements.
- The attributes that characterize the projects developed by the company.
- The rules that determine how to resolve process variability according to the values of the context attributes.

6.4.1.1 Organizational software process definition

Several Chilean software companies are already using EPF as the tool for defining their software process, even though there are other available tools [146]. It is a free tool that implements SPEM, the OMG process modeling standard notation. We will include this tool as part of our solution in order to enhance adoption. It allows use to define the software process breakdown, including primitives for specifying the variability of process element, as well as software process behavior using activity diagrams. The EPF allows exporting the software process as an XML file that is used as input for other activities. Moreover, EPF allows the formalization of:

- Software process breakdown, that is, all software process elements along with its relationships.
- Software process elements' variability, either optional or alternative. This is essential for identifying which elements should be resolved during process tailoring.
- Software process behavior using activity diagrams, even though this is not part of SPEM.

We consider the software process of Rhiscom for showing the formalization of software process in EPF. Figure 6.2 shows the software process breakdown that includes a formal specification of two optional elements and two alternative elements using SPEM primitives. Such a Figure also highlights the *Requirements* activity that is marked as optional in the last column.

Figure 6.3 shows the software process behavior of *Elaboration* phase and *Requirements* activity that include an specification of tailoring rules as annotations. We use red annotations for specifying the rules of optional elements and green annotations for specifying the rules of alternative elements.

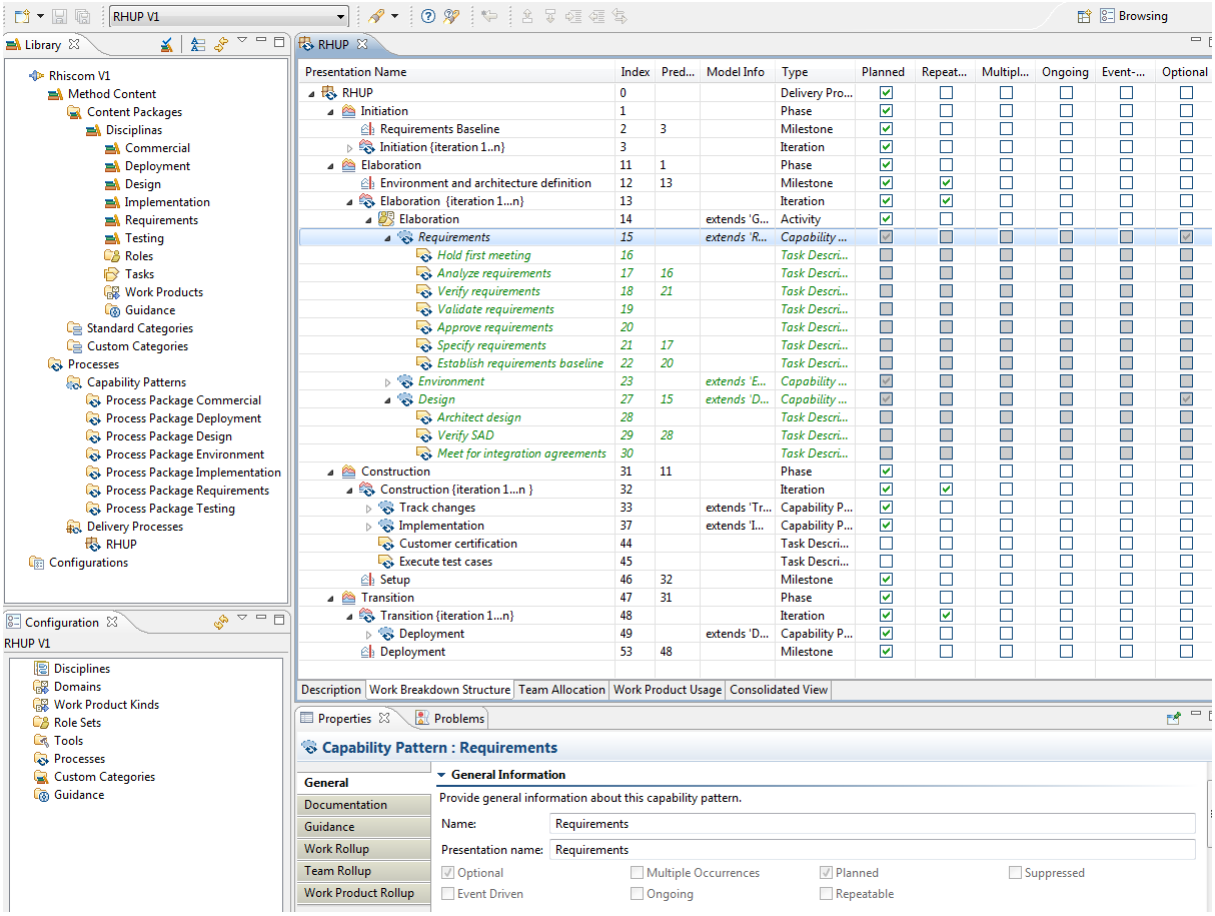
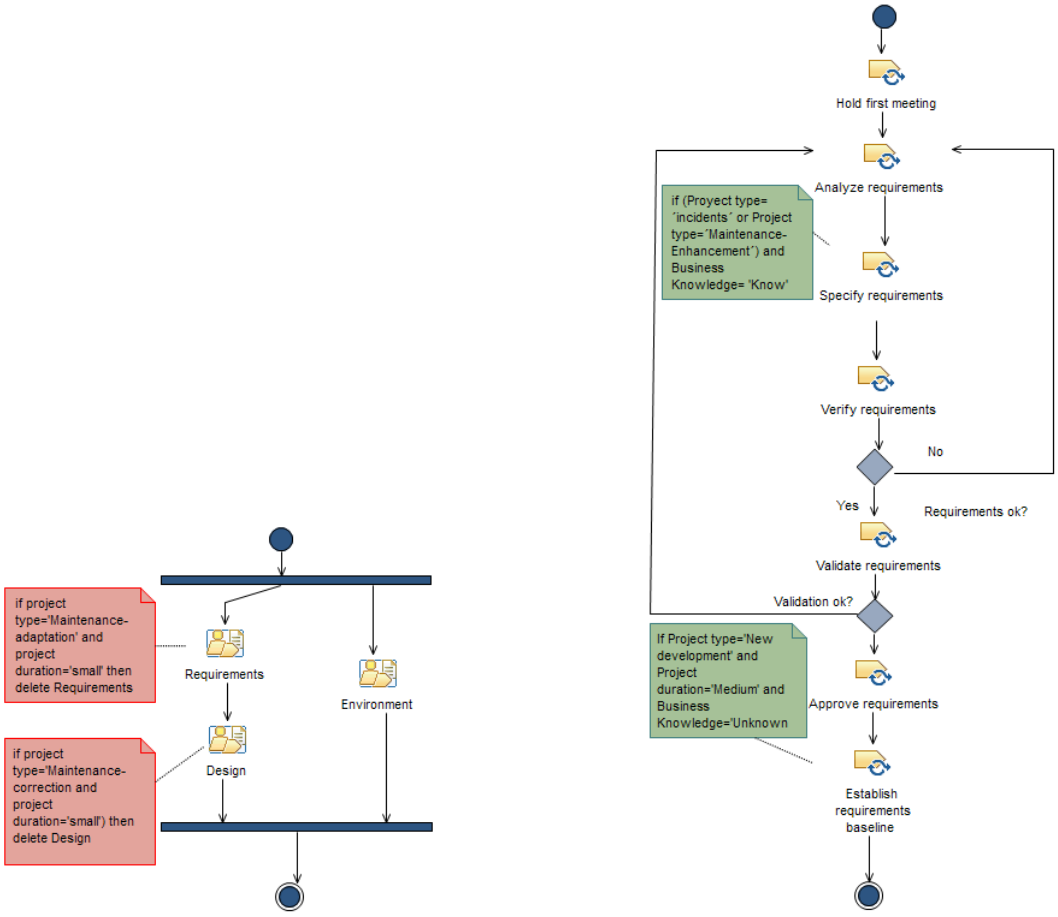


Figure 6.2: Rhiscom's software process breakdown in EPF.

6.4.1.2 Organizational context definition

The process engineer is in charge of defining which attributes of a project may affect how the process is tailored, as well as defining the potential values these attributes may take. In order to maximize usability, the tool provides a default definition of attributes organized into dimensions: Team, Project, Business, and Product dimensions. Within each of them there are attributes with default potential values. Figure 6.4 shows the attributes defined for the *Management* dimension along with their values. For example the attribute *Project type* may take values *Maintenance-correction*, *Maintenance-adaptation*, *Incidents*, *Maintenance-enhancement* and *New development*. The user interface allows editing this organizational context by adding or deleting attributes, and/or modifying their potential values. The output of this interface will be saved as the *Organizational Context Model*. The process engineer can edit the organizational context by adding, removing or editing dimensions, attributes and attribute values.

Finally, the *Organizational Context Model* is generated as XML file and is stored in a predetermined folder of ATAGeTT environment.



(a) Rhiscom's *Elaboration* Phase

(b) Rhiscom's *Requirements* Activity

Figure 6.3: Rhiscom's software process behavior in EPF.

6.4.1.3 Tailoring rules definition

Tailoring rules determine how variability is resolved during tailoring. Our integrated tool defines a user interface for defining tailoring rules that allows the process engineer to create models through a graphical environment. This graphical environment supports the definition of tailoring rules using our decision language for each variation point of the software process according to the values in the organizational context; therefore, the graphical environment considers the *Organizational Software Process* and the *Organizational Context Model*. The tailoring rules definition specify:

- The organizational software process model.
- Simple and/or complex rules for resolving each identified variable process element.
  1. Simple rules define a simple condition over a variable process element of the form if attribute = value then true/false, for optional elements. (e.g., Prototyping).
  2. Complex rules include conjunction or disjunction of simple rules. (e.g., Requirement Specification).
  3. For alternative elements, the result is the alternative to be included in the adapted software process when the condition holds.

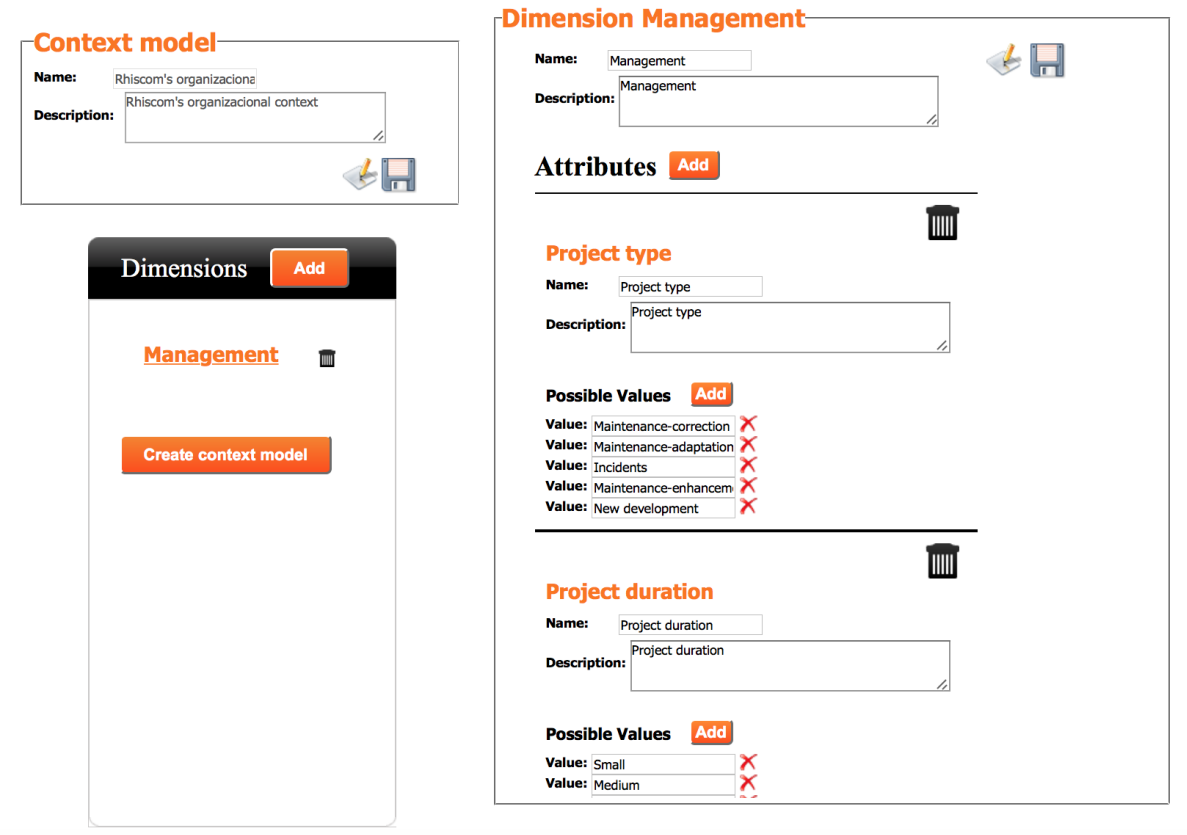


Figure 6.4: Organization context model definition of Rhiscom.

- The tailoring transformation using our HOT and ATL extractor.

This activity involves two steps, the definition of tailoring rules and the generation of tailoring transformations. In the first step, the tool guides the users by presenting them only the process elements defined as variable and allowing them specify a rule either simple or complex for each variation point. Figure 6.5 shows the selection of inputs for defining tailoring rules in XML format. The *Organizational Software Process* is the XML file exported from EPF that can be transformed in the *Organizational Software Process Model* using our *Injector*. The *Organizational Context* is the XML file generated by the *Organizational Context Definition*. Figure 6.6 shows the variable elements grouped in optional elements and alternative elements that are specified in the *Organizational Software Process Definition*. Finally, Figure 6.7 shows the definition of rules using the tailoring rules specification. The tool also counts on a menu where the context attributes and their potential values can be selected, not requiring any typing.

In the second step, the tool automatically generates the VDM that is the input of tailoring transformation generation. The tool executes the HOT and ATL extractor for automatically generating the tailoring transformation. Figure 6.8 shows an interface when the user writes a name that is used for automatically generating the VDM, the tailoring transformation model and the tailoring transformation.

Finally, the *Organizational Software Process Model*, the Variation Decision Model, the Tai-

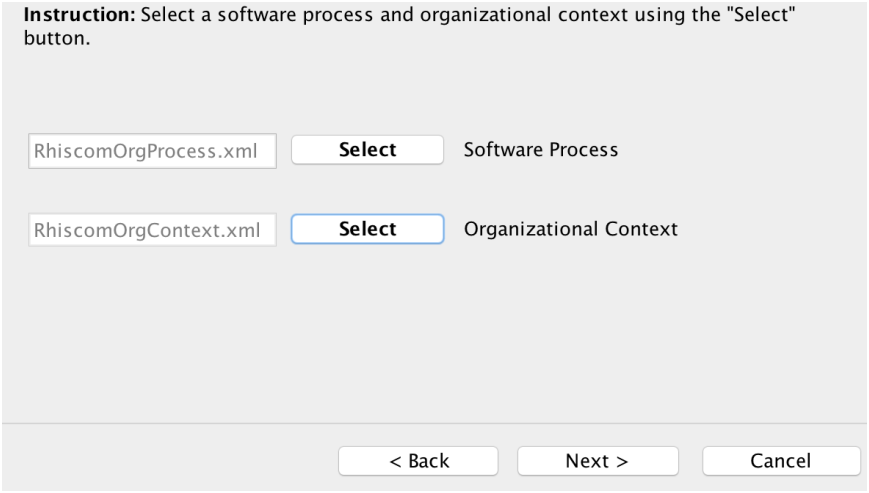


Figure 6.5: Selecting the organizational software process and the organizational context.

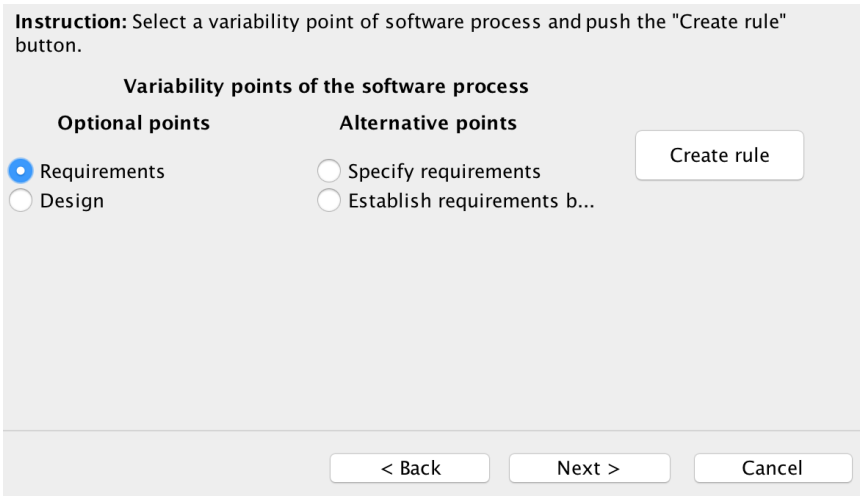


Figure 6.6: Selecting the variation elements specified in the organizational software process.

loring Transformation Model and the Tailoring Transformation are stored in a predetermined folder of ATAGeTT environment.

6.4.2 The project manager user interface

The process engineer is in charge of applying the software process tailoring, i.e., the process engineer defines a project context as a model and executes the tailoring transformation. Our tool integrates user interfaces for performing each of these activities in an automatic manner. The process engineer is the one who:

- Specifies the context of the particular project to be developed.
- Obtains and applies the adapted software process for executing the particular project.

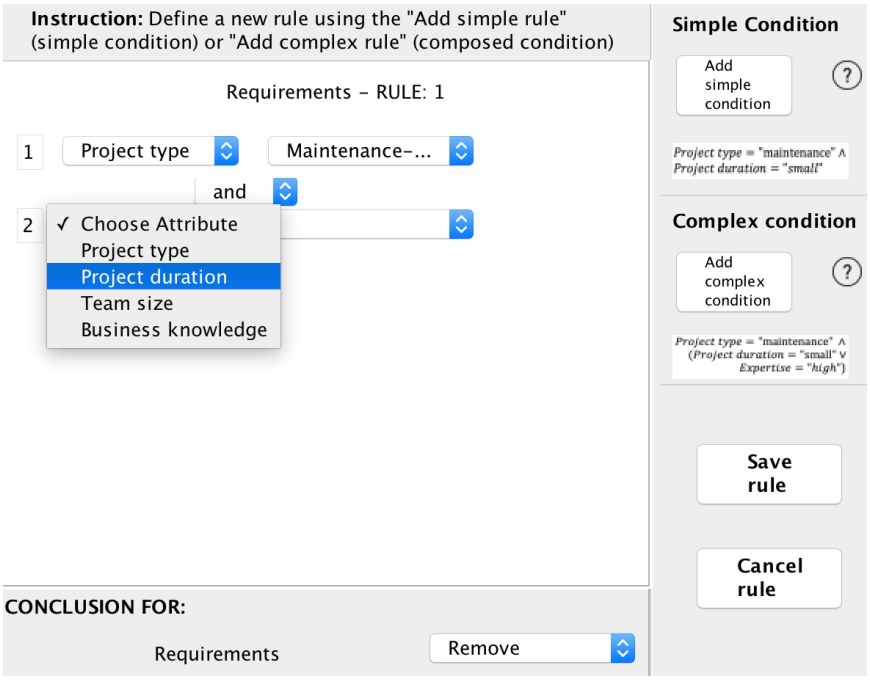


Figure 6.7: Defining tailoring rules for requirements activity.

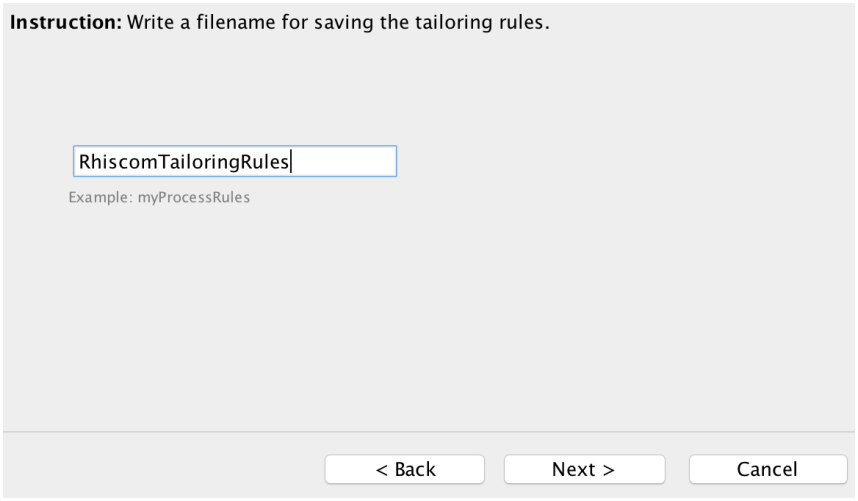


Figure 6.8: Executing the HOT and ATL extractor using the formal tailoring rules specification.

6.4.2.1 Project context definition

Our integrated tool allows the project manager to define the particular *Project Context* by setting the values of the *Organizational Context* for the project at hand, and then to obtain the *Adapted software Process*. To that end, the project manager can select the *Organizational Context* that was stored in a predetermined folder of ATAGeTT environment.

Figure 6.9 shows a *Project Context* where the project manager can specify the characteristics of a particular project. In this case, a maintenance project is defined using the following characteristics: the *Project Type* is *Maintenance-correction*, *Project duration* is *Small*, *Team*

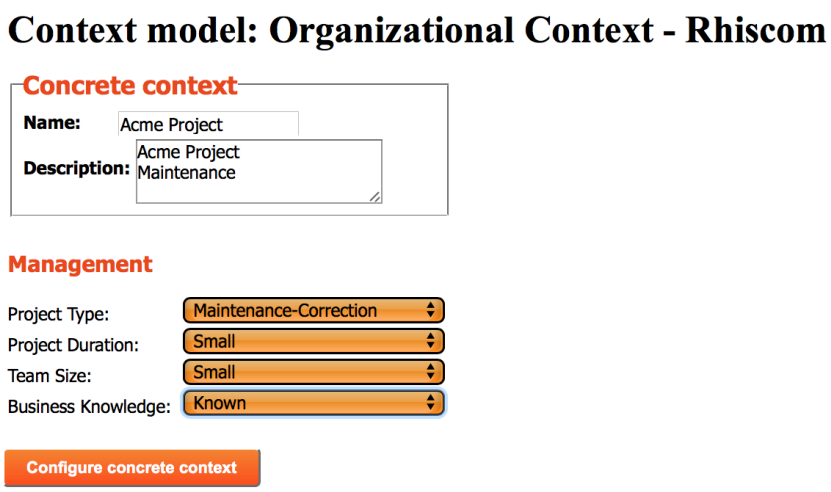


Figure 6.9: Project context definition

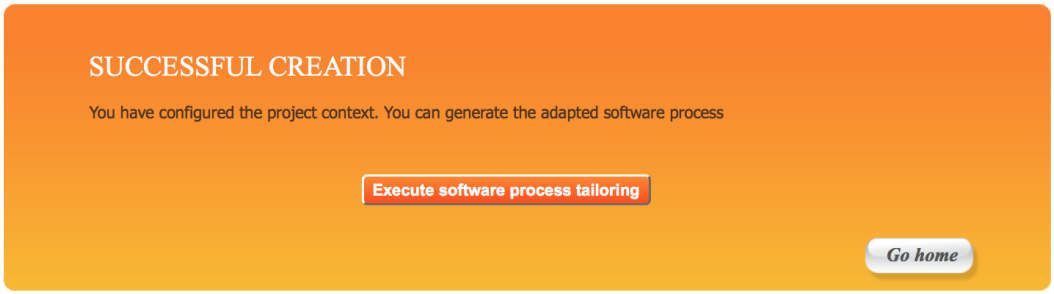


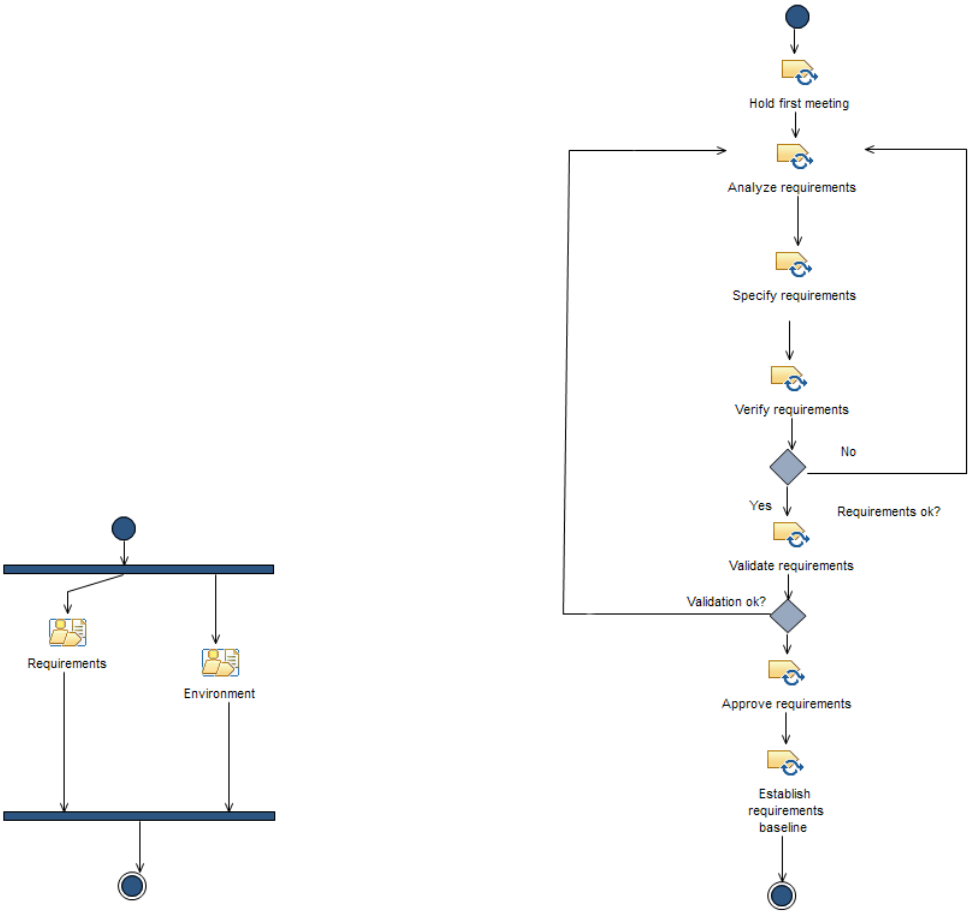
Figure 6.10: Adapted software process model generation

size is *Small* and *Business knowledge* is *Known*.

Once the project context is defined, the integrated tool executes the *Tailoring Transformation*. The tool automatically takes the *Project Context* model and the *Organizational Software Process* model as input, and generates the *Adapted Software Process* model. Notice that the *Tailoring Transformation*, the *Project Context* model and the *Organizational Software Process* model were stored in a predetermined folder of ATAGeTT environment.

Figure 6.10 shows a user interface that allows executing the tailoring transformation using our ATL launcher that applies the MDE-based software process tailoring automatically.

Finally, the *Organizational Software Process Model*, the Variation Decision Model, the Tailoring Transformation Model and the Tailoring Transformation Code are stored in a predetermined folder of ATAGeTT environment.



(a) Rhiscom's *Elaboration* Phase

(b) Rhiscom's *Requirements* Activity

Figure 6.11: The adapted software process of Rhiscom in EPF.

6.4.2.2 Adapted software process visualization

Also, the integrated tool allows the project manager to generate the *Adapted software Process* that can be visualized in EPF. To that end, the tool automatically takes the *Adapted software Process* model and generates the *Adapted software Process* as XML file using our *Extractor*. Figure 6.11 shows the *Adapted software process* in EPF. Notice that the *Design* activity is not part of the adapted software process.

Finally, the integrated tool allows potential users to define tailoring rules, generate tailoring transformations and apply software process tailoring without interacting with models, transformations and ATL code. In this sense, the MDE-based software process tailoring is transparent for the users and it is applied in a usable way.

Chapter 7

Exploratory Case Study

This chapter presents an evaluation of the suitability of applying our integrated tool for tailoring software processes in two software companies. The evaluation was conducted using an exploratory case study, where we compare the correctness and productivity of two different approaches: template-based tailoring and automatic MDE-based tailoring.

Section 7.1 introduces a motivation for applying the exploratory case study. Section 7.2 shows the design of the case study that considers a description of template-based and automatic MDE-based tailoring, the research question and a description of the cases. Section 7.3 presents the case study preparation. Section 7.4 explains the running of case study in two Chilean software companies. Section 7.5 shows the analysis of the gathered data. Section 7.6 presents the results and observations of the case study. Finally, Sec. 7.7 presents the lessons learned from this case study.

7.1 Motivation

There are several strategies for software process tailoring, in ADAPTE [117] and GEMS [117] projects, we found that most software companies use a template-based software tailoring strategy. The template-based software tailoring strategy [29] consists of a set of predefined software processes that are used in similar project contexts. On the other hand, as part of these research projects, we developed an MDE-based software process tailoring strategy [65] for generating adapted software processes to specific project contexts. In order to improve the adoption of MDE-based software process tailoring, we developed an integrated tool [143, 144] to support the MDE-based software process tailoring strategy (see Chapter 6). The automatic MDE-based software process tailoring allows users to generate models and transformations using an automatic strategy, and apply MDE-based software process tailoring in a usable manner.

A case study was designed for exploring the suitability of applying either the template-based or automatic MDE-based software process tailoring in terms of the correctness and productivity of each approach. The obtained results were then compared and discussed to

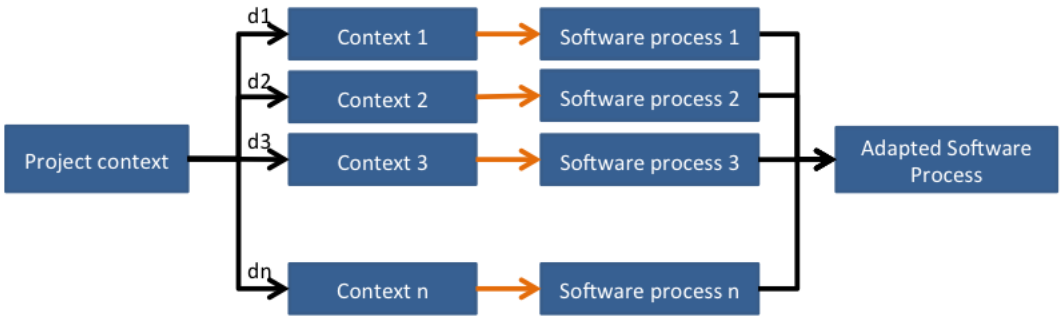


Figure 7.1: Template-based software process tailoring.

show the potential impact of the proposed integrated tool.

7.2 Case Study Design

We designed an exploratory case study where we evaluated the correctness and productivity of applying both, the template-based and also the automatic MDE-based software process tailoring strategy. In this sense, we followed the case study design proposed by Yin [173]. After presenting both strategies, we describe the evaluation strategy for comparing software processes, the characteristics evaluated in this case study, and the involved research question. Finally, we describe the case study itself.

7.2.1 Description of approaches

Software process tailoring refers to the activity of tuning a standardized process to meet the needs of a specific project [8]. There are several approaches for software process tailoring, ranging from always applying the same software process to defining a new software process for each project; the former is inefficient while the latter is expensive. Regardless this situation, there are two intermediate strategies used in industry and academia are template-based and automatic MDE-based software process tailoring. Next we briefly explain both approaches:

Template-based software process tailoring is a strategy that proposes to count on a series of predefined software processes for different project types, and choose the most appropriate of them for each project as shown in Fig. 7.1. It is the strategy followed by Crystal [29], that uses project criticality and team size as the project dimensions for determining the process to be applied, e.g., Clear, Yellow, Diamond, Quartz. This strategy inspires the template-based tailoring. Crystal, as well as all template-based tailoring by extension, are easy methodologies a company can adopt.

Automatic MDE-based software process tailoring applies an MDE approach for defining software process models, and adapting them to project contexts through transformations so that optimal processes are obtained [65, 89]. In order to automatically tailor

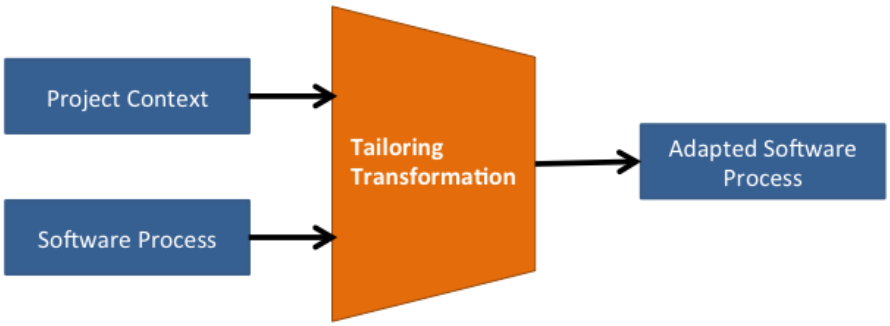


Figure 7.2: MDE-based software process tailoring

the software process it is necessary to formally define the project context, the organizational software process (including its potential variability) and the rules that make the transformation [65], as shown in Fig. 7.2. The model transformation resolves each variation point in the organizational process according to the context attribute values. Our integrated tool encapsulates this strategy [141].

7.2.2 Evaluation strategy for comparing the approaches

Software companies define their development processes in order to make them more predictable and manageable. However, once the process is defined, it is not useful for all kinds of projects since, e.g., large and complex projects require more activities and work products than small simple projects.

Template-based tailoring consists in having a series of predefined software processes and selecting the one to be applied in each project depending on the project context. On the other hand, automatic MDE-based software process tailoring generates the optimal software process for a project context. Notice that the two approaches require a project context for selecting one predefined software process or generating an optimal software process.

In the template-based tailoring, a particular project context is exactly the one defined for a predefined software process. However, different project contexts can select one of the predefined processes considering these project contexts. In this sense, the software process obtained by the template-based tailoring can be sub-optimal.

In order to evaluate the suitability of applying either the template-based and the automatic MDE-based software process tailoring, we compare the both approaches in terms of extra or missing software process elements. To that end, we consider two particular aspects: comparison between contexts and comparison between software processes.

Comparison between contexts allows computing *similarity* between a predefined project context and unexpected project context. *Similarity* is defined as the numerical value between zero and one, obtained by correspondingly comparing the context attributes values of two project contexts. This value is zero if all context attributes have different context attribute

values, whereas this value is one if all context attributes have the same context attribute values, and an intermediate value if some context attributes values are different. We assume that the project contexts are generated from the same organizational context.

However, when calculating the similarity we assume that all context information is equally relevant for the definition of a project, but our experience formalizing software processes indicates that this assumption is not true. A context attribute can be more relevant than others because there are tailoring rules that can remove software process elements. Notice that context attributes and context attribute values are part of the tailoring rules for selecting one predefined software process or generating an optimal software process.

The intuition of how a context attribute impact in the software process can be represented by the *weight* of such a context attribute. The *weight* of such a context attribute is defined as a value between zero and one, calculated as the relative occurrence frequency of the context attribute in the tailoring rules of the organizational process. This definition captures the impact of each context attribute for selecting one predefined software process.

Finally, we can define the comparison between contexts as *distance*. *Distance* is a numerical value that calculates *similarity* between two project contexts; it considers the *weight* of the context attributes.

Comparison between software processes allows computing *difference* between a predefined software process and an optimal software process. There are three concepts related to *difference*: *match*, *merge* and *diff*. *Match* refers to identifying how many and which software process elements coincide in the software process being compared. *Diff* consists in identifying those elements that are part of only of one software process. Merge two software process consists on building a new software process that includes all matching software process elements once, and also those software process elements in the difference. Both, *match* and *diff* can either yield a new software process or a measure of the value indicating to which degree both software processes match or differ.

To that end, we apply custom language-specific matching to compute the value of matching elements [104]. This approach allows users to define specific rules for computing the *match* and *diff* of two software processes in terms of models. In this sense, we can compute the difference between two software processes, using its models that are considered in the automatic MDE-based software process tailoring.

Finally, we can define the comparison between software processes as difference. Difference is a number of missing or extra software process elements that are computed using match and diff of software process models. In this case, we can use a particular tool called Eclipse Modeling Framework Compare (EMF Compare) [23]. EMF Compare allows comparing and merging generic models in a simple way.

7.2.3 Description of characteristics

The aim of the exploratory case study is evaluating the suitability of using MDE-based tailoring or template-based tailoring in terms of productivity and correctness. Several authors, such as Landauer et al. [93], Macleod et al. [96] and Jokela et al. [71], define some criteria for evaluating the productivity of any software product in terms of generating useful outputs. On the other hand, Macleod et al. [96] define correctness as goals achievement in a particular context. This meaning of correctness is similar to the ISO/IEC9126 [43] definition, in terms of achieving specific goals with accuracy and completeness in a specified context of use. Taking as reference these works about productivity and correctness, we consider the following evaluation criteria:

Productivity is measured as the existence of no extra work for executing a task. It implies that the productivity of the two approaches can be measured in terms of the missing and extra tasks.

Correctness is measured as the software process ability to perform the tasks exactly defined by the project context. In this case, the correctness can be evaluated by comparing the adapted software processes obtained using the two approaches.

7.2.4 Research question

We assume that, whenever the project context is exactly the one defined for a predefined software process, this software process is exactly the same as that automatically generated. However, for contexts different from those of predefined processes, how much are we losing or gaining by using predefined processes if compared to automatically tailored processes? In this sense, we define the following research question:

RQ1: Considering productivity and correctness, what is the difference between the tailored software processes obtained using each strategy?

7.2.5 Description of the cases

In this exploratory case study, two software companies were considered as independent cases. A summary of their characteristics appears in Table 7.1. Rhiscom[1] is an eighteen year-old software services and products company based in Santiago, Chile. It develops point of sale software for the retail industry. Mobius[2] is a five year-old software services, products and hardware company also based in Santiago, Chile. This company develops attendance management software and video-conference services. Concerning the type of services, they offer: a) projects (software development); b) products (COTS application development); and c) services (consulting, training, etc.). Concerning to the number of long-term clients, Rhiscom

[1]Rhiscom website: http://www.rhiscom.com
[2]Mobius website: http://www.mobius.cl

Table 7.1: Characteristics of the software companies.

Company	**Business (activity)**	**Age (years)**	**Clients**	**Total (people)**	**People in software development tasks**	**People in other tasks**
Rhiscom	Retail (point of sale software)	18	>20	51	43 (84%)	8 (16%)
Mobius	Attendance management and video-conference service	5	6	25	11 (44%)	14 (56%)

has 20 clients and Mobius has 6 clients. Both companies do not have ISO certifications, but they are working to apply for obtaining it in the short-term.

Considering their size in terms of number of employees, Mobius is a small software company (25 people), and Rhiscom is a medium-sized company (51 people). Moreover, Rhiscom sells its products and services in 5 other countries. Table 7.1 also shows the number of people working in tasks specifically related to software development (e.g., project managers, developers, testers, analysts, etc.) and people working in other tasks (e.g., secretary, sales managers, cleaning staff, etc.). There we can see that Rhiscom has relatively more employees committed to software engineering tasks (86%) than Mobius (44%).

7.3 Case Study Preparation

The units of analysis for this exploratory case study were the organizational software process with variability, and the adapted software processes (that includes all the process elements that are needed for a specified project). Next we explain the requirements of the tailoring strategies:

- Template-based tailoring strategy requires predefined software processes and their corresponding project context. Depending on the characteristics of the project to be addressed, there is a particular predefined process that is the most appropriate.
- Automatic MDE-based tailoring strategy requires the organizational project context (that describes attributes and their values for defining a project context), project contexts (that describes characteristics of a particular project), organizational software process (that describe activities, roles, tasks, work products for conducting a software development) and tailoring rules (that describe how to tailor the organizational software process for a particular project context).

Both companies satisfy the requirements for each tailoring strategy: the organizational software process with variability, the organizational context, the tailoring decisions, the predefined software processes and its project contexts. Both companies have all requirements for each strategy. In the next subsections, we present the requirements of each company.

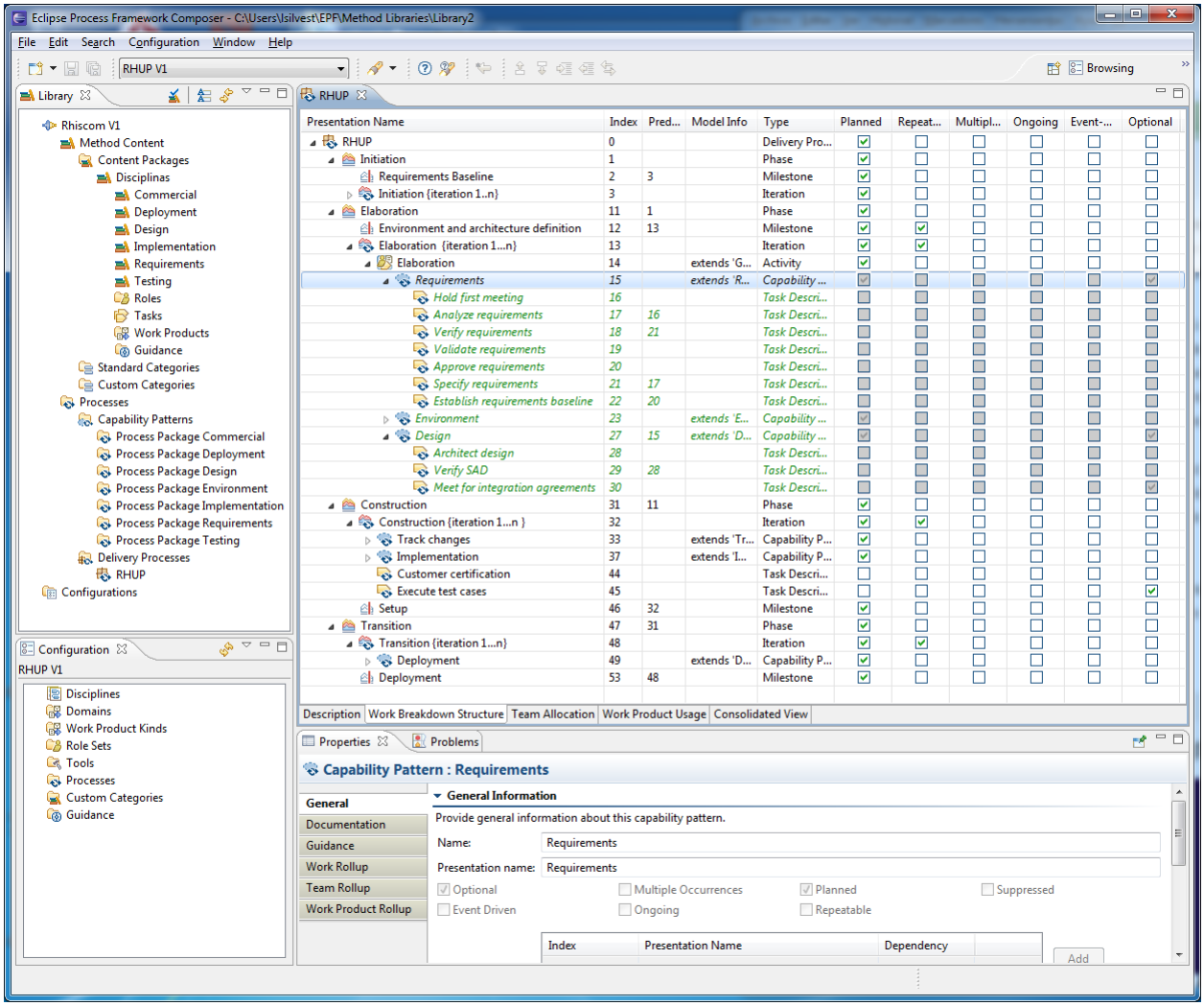


Figure 7.3: Software process breakdown of Rhiscom that includes four optional elements.

7.3.1 Rhiscom

Figure 7.3 shows a snapshot of the Rhiscom's organizational software process in EPF. This process includes information about the tasks, roles and work products related to initiation, elaboration, construction and transition. It also includes information about the variabilities of the process.

Rhiscom's organizational software process has six variable elements: four optional elements and two alternative elements that can be reused in different parts of the software process. Figure 7.3 highlights *Requirements* capability pattern as an *Optional* element and it is marked in the last column. On the other hand, alternative elements are specified using *replaces*; i.e., an alternative element can be replaced by a base element that is specified as part of the method content. For instance, *Specify requirements* is a base element that can be replaced by *Specify requirements in plain text* or *Specify requirements in use cases*.

Table 7.2 shows Rhiscom's organizational context. It has five attributes: *Project type*, which has five potential values: *New development*, *Maintenance-enhancement*, *Maintenance-*

Table 7.2: Rhiscom's organizational context

Dimensions	**Attributes**	**Values**
Management	Project type	New development Maintenance-enhancement Maintenance-correction Maintenance-adaptation Incidents
	Project duration	Small Medium Large
	Team size	Small Medium Large
	Business knowledge	Known Affordable Unknown

Table 7.3: Rhiscom's predefined contexts

	Management													
	Project type					Project duration			Team size			Business knowledge		
	New development	Maintenance-enhancement	Maintenance-correction	Maintenance-adaptation	Incidents	Small	Medium	Large	Small	Medium	Large	Known	Affordable	Unknown
New project	√							√		√				√
Corrective project			√			√			√			√		
Incident project					√	√			√			√		

correction, *Maintenance-adaptation* and *Incidents*; *Project duration* that has three potential values: *Small*, *Medium*, *Large*; *Team size* that has five potential values: *Small*, *Medium*, *Large*; and *Business knowledge* that has three potential values: *Known*, *Affordable*, *Unknown*. These context attributes are grouped in one dimension for better comprehension: *Management*.

Table 7.3 shows three predefined project contexts, corresponding to typical kinds of projects: *New project*, *Corrective project* and *Incident project*. Their corresponding contexts are defined by particular combinations of organizational context attributes values. For each of these context, there is a predefined software process that will be applied whenever the new project context matches the predefined context. For example, a predefined context for a *New project* is defined as: *Project type* is *New development*, *Project duration* is *Large*, *Team size* is *Medium* and *Business Knowledge* is *Unknown*. Moreover, its predefined software process is the same organizational software process (see Fig. 7.3), because all software process elements are required and the alternative software process elements are resolved.

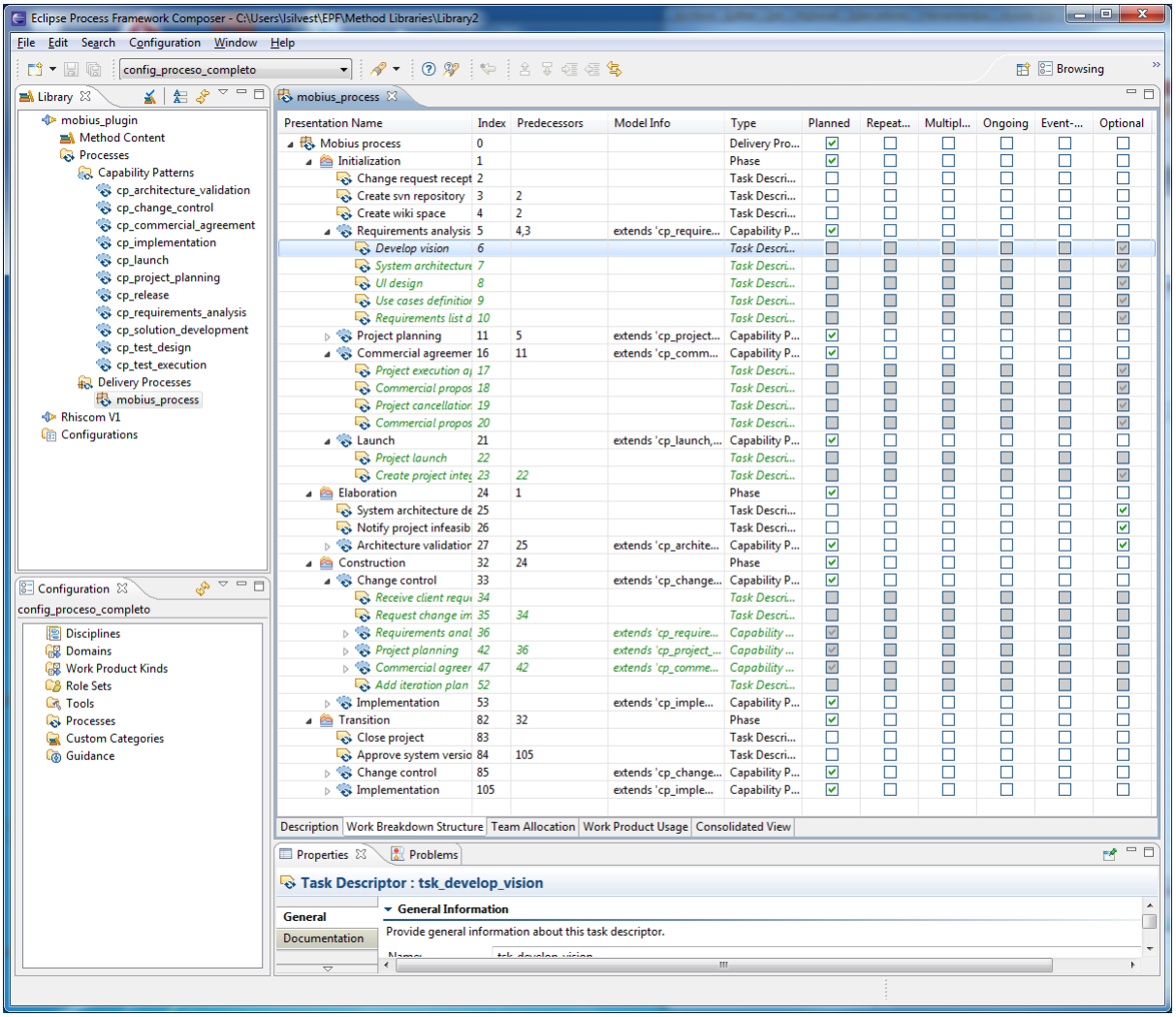


Figure 7.4: Software process breakdown of Mobius that includes sixteen optional elements.

7.3.2 Mobius

Figure 7.4 shows a snapshot of the Mobius' organizational software process. The organizational software process includes information about the tasks, roles and work products related to initiation, elaboration, construction and transition. It also includes information about the variabilities of the process.

Mobius' organizational software process has sixteen variable elements: sixteen optional elements. Figure 7.4 highlights *Develop vision* task as optional element (marked in the last column). In this case, Mobius's organizational software process does not have alternative elements.

Table 7.4 shows Mobius' organizational context, which has ten attributes. *Project type* has three potential values: *New development*, *Corrective*, *Non-corrective*. *System type* has two potential values: *Guarantee*, *No guarantee*. *Interaction with other system* has two potential values: *Simple*, *Complex*. *Team experience* has two potential values: *Yes*, *No*. *Usability* has two potential values: *High*, *Low*. *Documentation* has two potential values: *Report*, *No report*.

Table 7.4: Mobius' organizational context

Dimensions	Attributes	Values
Project	Project type	New development Corrective Non-corrective
System	System type	Guarantee No guarantee
	Interaction with other systems	Simple Complex
Team	Team experience	Yes No
Product	Usability	High Low
	Documentation	Report No report
	Complexity of the functionality	High Average Low
	Data access layer	Yes No
	Business logic	Yes No
	Source code	Yes No

Complexity of the functionality has three potential values: *High*, *Average*, *Low*. *Data access layer* has two potential values: *Yes*, *No*. *Business logic* has two potential values: *Yes*, *No*. *Source code* has two potential values: *Yes*, *No*. These context attributes are grouped in four dimensions: *Project* (project type), *System* (system type, interaction with other systems), *Team* (team experience), *Product* (usability, documentation, complexity of the functionality, data access layer, business logic, source code).

Table 7.5 shows three predefined project contexts, corresponding to typical kinds of projects: *New project*, *Corrective project* and *Non-corrective project*. For each of them there is a predefined software process that will be applied whenever the new project context matches the predefined context. For example, a predefined context is *New project* that is defined from project context: *Project type* is *New development*, *System type* is *Guarantee*, *Interaction with other systems* is *Complex*, *Experience in the architecture* is *No*, *Usability* is *High*, *Documentation* is *No report*, *Complexity in the functionality* is *High*, *Data access layer* is *Yes*, *Business logic* is *No* and *Source code* is *No*, and its predefined software process is the same organizational software process (see Fig. 7.4) because all software process elements are required.

7.4 Case Study Execution

In this exploratory case study, we conduct a focus group as the method for data collection. From August 2013 to January 2014, focus groups were conducted in two software companies.

Table 7.5: Mobius' predefined contexts

	Project			**System**				**Team**	
	Project type			System type		Interaction with other systems		Experience in the architecture	
	New development	Corrective	Non-corrective	Guarantee	No guarantee	Simple	Complex	Yes	No
New project	√			√			√		√
Corrective project		√			√	√		√	
Non-corrective project			√	√		√		√	

	Product												
	Usability		Documentation		Complexity in the functionality			Data access layer		Business logic		Source code	
	High	Low	Report	No report	High	Average	Low	Yes	No	Yes	No	Yes	No
New project	√			√	√			√			√		√
Corrective project		√	√			√			√	√			√
Non-corrective project	√			√			√		√		√	√	

Table 7.6: Summary of sessions

Company	Date	Sessions	Hours (total)	Company team	Research team	Activities	Tools
Rhiscom	August 2013	3	7	2	2	– Review organizational software process – Review tailoring decisions – Define experimental project contexts – Apply tailoring strategies	EPF
Mobius	December 2013 - January 2014	3	7	2	2		

Both companies formalized their descriptive software process with variability in EPF as part of the ADAPTE project.

Table 7.6 shows dates, activities, tools, sessions, hours and team in the case study for each company. The study considers four main activities: review organizational software process for updating software process elements, identify tailoring decisions as annotations in the software process, define project contexts for characterizing particular projects and apply tailoring strategies.

The focus groups lasted for seven hours in total and involved three sessions (between 2 and 3 hours each). Each session was conducted in the company's offices with both, the company and the research teams. In the Rhiscom study, a total of four people participated during the focus group sessions: two people from the software company team (analyst and quality assurance leader) and two people from the research team (the author of this thesis and other PhD. student). Similarly, in the focus group sessions in Mobius also participated four people: two people (project manager and quality assurance leader) from the software company and two people from the research team (the author of this thesis and a MSc. student). In the following subsections we present the activities that were conducted in each company.

7.4.1 Review of the organizational software processes

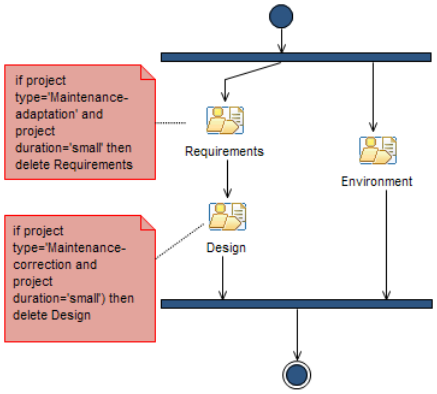
Rhiscom's organizational software process was formalized in 2011 and it had four variable elements: two optional and two alternative elements. On the other hand, Mobius's organizational software process was formalized in 2012 and it had sixteen variable elements: all are optional. In order to verify these software processes, we reviewed the organizational software processes with project managers of the corresponding company.

In the first session, the research and company team reviewed and verified the organizational software processes. Table 7.7 shows a summary of organizational software process reviewed in both software companies. Rhiscom's organizational software process did have important changes and had six variable elements: four optional elements (includes 2 new optional elements) and two variable process elements. On the other hand, Mobius' organizational

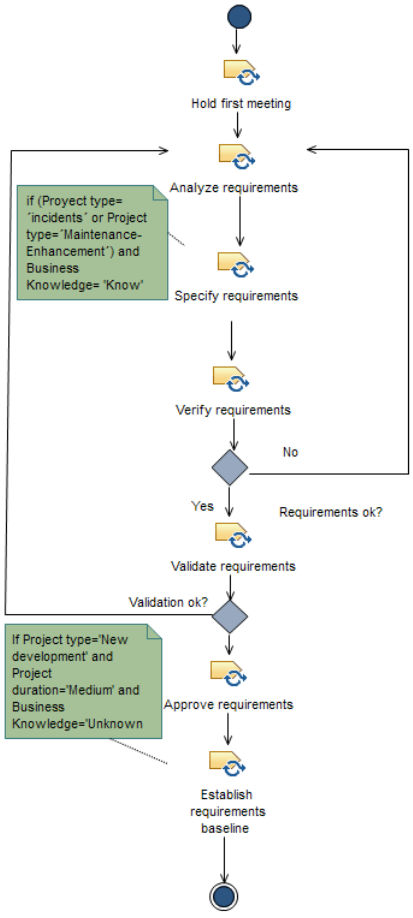
Table 7.7: Rhiscom's organizational software process reviewed.

Company	Software process	Tasks	Roles	Work products	Variable elements
Rhiscom	RUP	36	10	37	6
Mobius	OpenUP	47	10	44	16

software process did not have any change in the variable elements and had the same sixteen variable process elements.



(a) A specification as annotation of an optional element in *Construction* phase.



(b) A specification as annotation of an alternative element in *Requirements* activity.

Figure 7.5: Part of tailoring decisions in Rhiscom's organizational software process.

7.4.2 Identification of tailoring decisions

Tailoring decisions of Rhiscom and Mobius were defined in 2011 and 2012, respectively. The tailoring decisions were specified as annotations in the organizational software process for each variable element. To verify these decisions, we reviewed them in each software company.

In the first session, the research and company teams reviewed tailoring decisions. We

Table 7.8: Tailoring decisions for variable elements of Rhiscom's software process.

Decision	Process Element	Variability Type	Conditions	Conclusion
Decision 1	Requirements	Optional	(Project type= Maintenance-adaptation) AND (Project duration=Small) AND (Business knowledge=Known)	Remove: Requirements
Decision 2	Design	Optional	(Project type= Maintenance-correction) AND (Project duration=Small) AND {(Team size=Small) OR (Business knowledge=Know)}	Remove: Design
Decision 3	Execute Test Cases	Optional	(Project type= Maintenance-correction) AND {(Project duration=Small) OR (Business knowledge=Affordable)}	Remove: Execute Test Cases
Decision 4	Meet for integration agreements	Optional	(Business knowledge=Known) AND (Project type= Maintenance-enhancement)	Remove: Meet for integration agreements
Decision 5	Specify Requirements	Alternative	{(Project type=Incidents OR Project type= Maintenance-enhancement)} AND (Business knowledge=Know)	Replace for: Specify Requirements in plain text
Decision 6	Establish Requirements Baseline	Alternative	Project type=New development AND Project duration=Medium AND Business knowledge=Unknown	Replace for: Establish Requirements Baseline and Test Cases

defined red notes for describing optional elements and green notes for alternative elements. Figure 7.5a shows a snapshot of Rhiscom's software process in EPF. In the software process, we can see annotations that describe conditions and conclusions as text for each variable element. For example, Fig. 7.5a shows a red note for describing optional decisions for *Requirements* and *Design* activities. Figure 7.5b shows green notes for describing alternative decisions to *Specify requirements* and *Establish requirements baseline* tasks.

In order to improve the description of tailoring decisions, we specified them as a table that includes information as: Process element, Variability type, Conditions and Conclusions. Table 7.8 shows a description of six tailoring decisions.

Concerning Mobius, Figure 7.6 shows a snapshot of the organizational software process in EPF that also includes tailoring decisions as annotations. For example red notes for

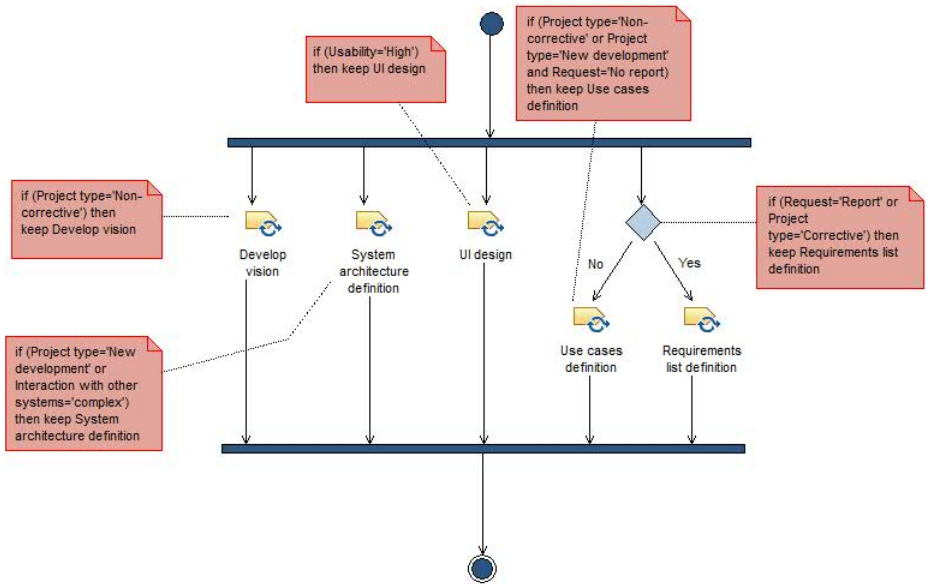


Figure 7.6: Part of tailoring decisions in Mobius's organizational software process.

describing tailoring decisions for *Develop vision*, *System architecture_ definition*, *UI design*, *Use cases definition*, *Requirements list definition* tasks. Table 7.9 shows a description of all sixteen tailoring decisions.

Table 7.9: Decisions for Mobius' variable process elements.

Decision	**Process Element**	**Variability Type**	**Conditions**	**Conclusion**
Decision 1	tsk_ develop_vision	Optional	(Project type=Non-corrective)	Remove: Requirements
Decision 2	tsk_ system_arq_def	Optional	(Project type=New development) OR (Interaction with other systems= Complex)	Remove: Design
Decision 3	tsk_ ui_design	Optional	(Usability=High)	Remove: Design
Decision 4	tsk_ use_cases_def	Optional	{(Project type=Non-corrective) OR (Project type=New development)} AND (Documentation=No report)	Remove: Design
Decision 5	tsk_ req_lists_def	Optional	(Documentation=Report) OR (Project type=Corrective)	Remove: Design
Decision 6	tsk_ commercial_ proposal_def	Optional	(Project type=Non-corrective) OR {(Project type=Corrective) AND (System type=No guarantee)}	Remove: Design

Decision 7	tsk_ commercial_ proposal_ validation	Optional	(Project type=Non-corrective) OR {(Project type=Corrective) AND (System type=No guarantee)}	Remove: Design
Decision 8	tsk_ project_ execution_ approval	Optional	(Project type=Non-corrective)	Remove: Design
Decision 9	tsk_project_ cancellation_ notification	Optional	(Project type=Non-corrective)	Remove: Design
Decision 10	tsk_create_ project_ integration_ server	Optional	(Source code=Yes)	Remove: Design
Decision 11	cp_ architecture_ validation	Optional	{(Project type=New development) OR (Project type=Non-corrective)} AND (Experience in the architecture= No)	Remove: Design
Decision 12	tsk_ notify_project_ infeasibility	Optional	{(Project type=New development) OR (Project type=Non-corrective)} AND (Experience in the architecture= No)	Remove: Design
Decision 13	tsk_ perform_ detailed_design	Optional	(Complexity of the functionality= High)	Remove: Design
Decision 14	tsk_integrate_ software	Optional	(Source code=Yes)	Remove: Design
Decision 15	tsk_create_ unit_tests	Optional	(Business logic=Yes) OR (Data access layer=Yes)	Remove: Design
Decision 16	tsk_execute_ unit_tests	Optional	(Business logic=Yes) OR (Data access layer=Yes)	Remove: Design

7.4.3 Defining the project contexts

In the second session, the project contexts were defined for applying template-based and automatic MDE-based software process tailoring. These contexts were based on the organizational contexts of Rhiscom and Mobius, respectively. In both cases, five experimental project contexts were designed as contexts that can be feasible in the business activities of each company. Teams from the university and the company participated in this activity.

Table 7.10 shows five experimental project contexts of Rhiscom. Table 7.10 highlights *Context A* for characterizing new projects: *Project type* is *New development*, *Project duration* is *Medium*, *Team size* is *Medium* and *Business knowledge* is *Affordable*.

Table 7.10: Rhiscom's experimental project contexts.

	Management													
	Project type					Project duration			Team size			Business knowledge		
	New development	Maintenance-enhancement	Maintenance-correction	Maintenance-adaptation	Incidents	Small	Medium	Large	Small	Medium	Large	Known	Affordable	Unknown
Context A	√						√			√			√	
Context B		√					√			√		√		
Context C			√			√			√				√	
Context D				√		√			√				√	
Context E					√	√				√				√

Table 7.11 shows experimental project contexts of Mobius. For example, *Context B* is a context for characterizing corrective projects: *Project type* is *Corrective*, *System type* is *Guarantee*, *Interaction with other systems* is *Complex*, *Experience in the architecture* is *No*, *Usability* is *Low*, *Documentation* is *Report*, *Complexity in the functionality* is *Average*, *Data access layer* is *No*, *Business logic* is *No* and *Source code* is *Yes*.

7.4.4 Applying both tailoring strategies

In the third session, we applied the template-based and the automatic MDE-based tailoring strategies. We also provided tables to compare the predefined context and the experimental context. This was required for selecting the most appropriate predefined process, i.e., the one that is *closest* to the experimental context. Once the closest context is selected, we analyzed the resulting template-based process so that we can compare it with the automatically tailored software process. For the predefined contexts, we assumed that the corresponding process in the template-based strategy is exactly the same as that obtained with automatic MDE-based tailoring.

Table 7.12 shows the number of tasks and the number of variation points in each process. We can see that Mobius's software process is more detailed, since it includes more than twice the number of tasks, if compared with Rhiscom's software process. However, in both software processes, the number of variation points is about one fourth the number of tasks.

We computed the similarity between each of the five experimental contexts with respect to predefined contexts project using EMF Compare[3]. We call *Distance* to the measure of similarity. The *Distance* between two context instances is calculated as follows: first we consider the weight for each context variable, and then the *Distance* between two contexts is

[3]EMF Compare - http://www.eclipse.org/emf/compare/

Table 7.11: Mobius' experimental project contexts.

	Project			**System**				**Team**	
	Project type			System type		Interaction with other systems		Experience in the architecture	
	New development	Corrective	Non-corrective	Guarantee	No guarantee	Simple	Complex	Yes	No
Context A	√				√	√			√
Context B		√		√			√	√	
Context C			√	√		√		√	
Context D			√		√		√	√	
Context E		√		√			√		√

	Product												
	Usability		Documentation		Complexity in the functionality			Data access layer		Business logic		Source code	
	High	Low	Report	No report	High	Average	Low	Yes	No	Yes	No	Yes	No
Context A		√	√		√			√		√		√	
Context B		√	√			√			√		√	√	
Context C	√		√		√				√		√		√
Context D	√		√			√		√		√			√
Context E		√		√			√	√		√		√	

Table 7.12: Size of software processes.

Company	**Number of Tasks**	**Variation Points**
Rhiscom	48	10
Mobius	104	26

Table 7.13: The weights of Rhiscom's context attributes.

Variables	Weight	Values
Project type	0.35	New Development Maintenance-Enhancement Maintenance-Correction Maintenance-Adaptation Incidents
Project Duration	0.24	Small Medium Large
Business Knowledge	0.35	Known Affordable Unknown
Team Size	0.06	Small Medium Large

Table 7.14: Similarities between predefined and experimental contexts in Rhiscom.

Experimental Contexts	Predefined Contexts		
	New project	Corrective project	Incident project
A	0.41	0.00	0.00
B	0.06	0.35	0.35
C	0.00	0.65	0.30
D	0.00	0.30	0.30
E	0.41	0.24	0.59

the sum of the weights of those variables in each context that match. A higher value implies higher similarity between contexts.

7.4.4.1 Tailoring in Rhiscom

Table 7.3 described the predefined project contexts considered for Rhiscom (New project, Corrective project and Incident project). Table 7.13 describes Rhiscom's project context attributes as well as their weight and potential values. The weight of the context variables is considered as the relative frequency they affect process elements variability, i.e., the more variation point conditions the attribute participates in, the higher the weight of the attribute. The sum of the weight of all attributes is 1.

Table 7.14 shows the distance between predefined contexts: New project, Corrective project and Incident project, with the five experimental ones. We have highlighted those closest contexts. There are two experimental contexts –B and D– that are equidistant from two existing ones: Corrective project and Incident project for B, and Corrective project and Incident project for D. We use these contexts for illustrating both tailoring strategies.

```
--Variability point: Establish Requirements Baseline
helper def: alternativeRule(tu:MM!TaskUse): MM!TaskDefinition =
if(Sequence{'Establish Requirements Baseline',
   'Specify Requirements'}.includes(tu.name))}
   then(
       if('Specify Requirements'= tu.name)
           then thisModule.ruleAlt2(tu)
           else (
           if('Establish Requirements Baseline'= tu.name)
               then thisModule.ruleAlt1(tu)
               else tu.linkTask
           endif
           ) endif )
   else tu.linkTask
endif;
```

```
--Alternative Rule for Establish Requirements Baseline
helper def:ruleAlt1(tu:MM!TaskUse): MM!TaskDefinition=
   if(thisModule.getValue('Project Type') =
                          'New Development' and
         thisModule.getValue('Project Duration') =
                          'Medium' and
         thisModule.getValue('Business Knowledge') =
                          'Unknown') then
         thisModule.getTaskDefinition
         ('Establish Requirements Baseline and Test Cases')
   else thisModule.getTaskDefinition(tu.name)
endif;
```

Figure 7.7: Rhiscom's automatically tailoring ATL transformation.

Automatic MDE-based Tailoring

The automatic tailoring executes ATL rules for generating the adapted software process. Figure 7.7 shows an excerpt of the transformation applied to the software process in Fig. 7.5b. Figure 7.8 shows the automatically tailored *Requirements* activity for context E.

In this case we can see that the *Specify requirements in plain text* has been selected and not the *Specify requirements in use cases*. Similarly, the *Requirements* activity has kept the *Establish requirements baseline without test cases* and not the *Establish requirements baseline and test cases*.

Template-based Strategy

For the template-based strategy, Corrective project and Incident project are equally close to B. We show the *Requirements* activity predefined for contexts Corrective project and Incident project in Figs. 7.9a and 7.9b, respectively. We can see that all decisions about alternatives have been resolved and the resulting process for context C is exactly the same as that automatically tailored, i.e., there are no extra nor missing tasks, and no different alternative task chosen either; however the *Requirements* activity tailored for context Corrective project chooses the *Specify requirements in use cases* exhibits no missing nor extra tasks, but it chooses a different alternative: while the automatically tailored activity chooses *Specify requirements in plain text*, the template-base tailored to context Corrective project exhibits *Specify requirements in use cases*.

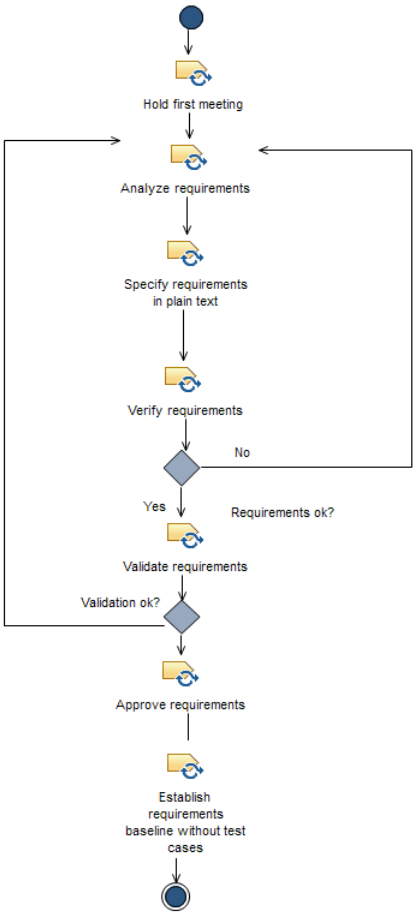


Figure 7.8: Rhiscom's *Requirements* activity automatically tailored to context E.

7.4.4.2 Tailoring in Mobius

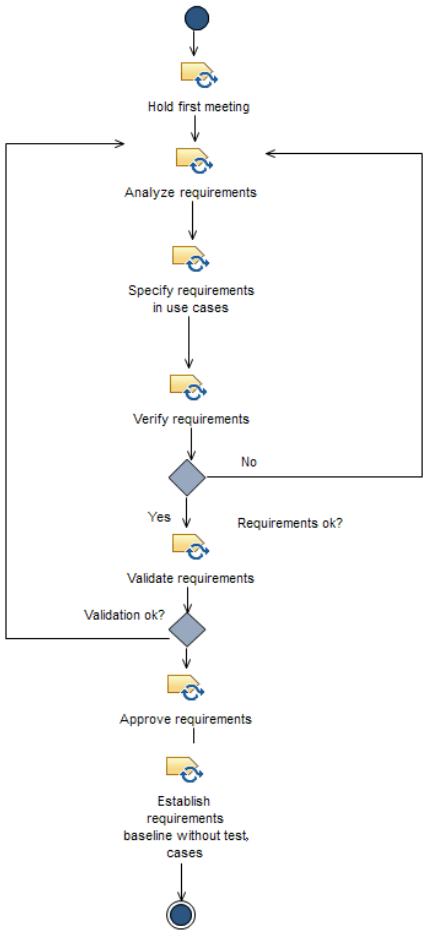
Figure 7.5 describes the predefined project contexts considered for Mobius (New project, Corrective project and Non-corrective project) as well as five other experimental project contexts that are different from the former ones.

Table 7.15 describes Rhiscom's project context attributes as well as their weight and potential values.

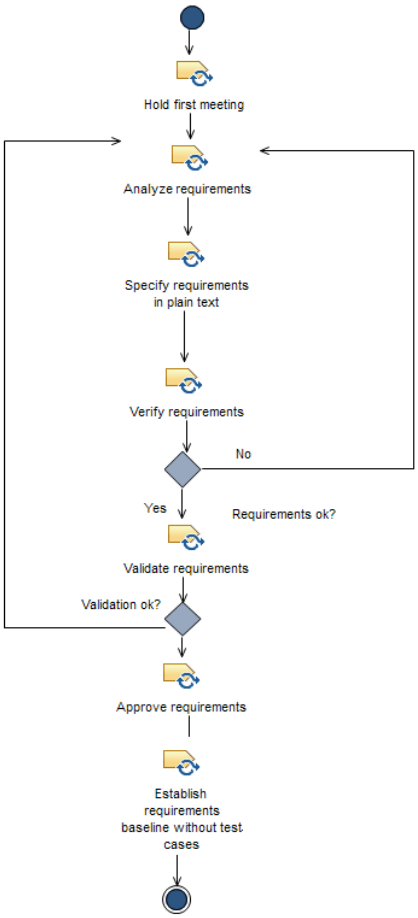
In Tab. 7.16 we report the *Distance* between predefined and experimental contexts. In each case we highlighted the closest context. Notice that E es equally distant from New project and Corrective project. We will use this context as an example to illustrate the consequences of applying the template-based strategy depending on the process we select to apply.

Automatic MDE-based Tailoring

The automatic tailoring executes ATL rules for generating the tailored process so that all variation points are resolved. Figure 7.10 shows an excerpt of the transformation applied to the software process in Fig. 7.6. We can see in the highlighted rule that *System architecture*



(a) Rhiscom's *Requirements* activity tailored to context corrective project.



(b) Rhiscom's *Requirements* activity tailored to context incident project.

Figure 7.9: Predefined software process for experimental Context B.

definition is included whenever *('Project type' = 'New development' or 'Interaction with other systems' = 'Complex'*, as is the case for context H where even though *'Project type'* is *"Corrective'*, *'Interaction with other systems'* is *'Complex'* and therefore the whole disjunction is *'true'*. To exemplify how automatic tailoring computes the tailored process, we show in Fig. 7.11 the result of automatically tailoring Mobius's *Requirements* activity to context E. We can see that when considering the alternative condition, it evaluates to *'Yes'* because *'Project type' = 'Corrective'*, and therefore the alternative *Requirements list definition* is chosen.

Template-based Strategy

When comparing context E with the predefined New project, Corrective project or Non-corrective project, we realized that contexts Corrective project or Non-corrective project are equidistant from E (see Tab. 7.16). Therefore we show both processes, one corresponding to context New project in Fig. 7.12a and another corresponding to context Corrective project in Fig. 7.12b. We can see that both results are different between them, and they are also different from that tailored automatically (see Fig. 7.11). While the template-based tailored *Requirements* activity to context New project in Fig. 7.12a has one extra task -*UI design*-,

Variables	Weight	Values
Project type	0.30	New Development Corrective Non Corrective
System Type	0.10	Guarantee No Guarantee
Interaction with other systems	0.10	Simple Complex
Team Experience	0.10	Yes No
Usability	0.10	High Low
Documentation	0.05	Report No report
Functionality Complexity	0.05	High Average Low
Data Access Layer	0.05	Yes No
Business Logic	0.05	Yes No
Source Code	0.05	Yes No

Table 7.15: Mobius's organizational context model.

Table 7.16: Similarities between predefined and experimental contexts in Mobius.

Experimental Contexts	Predefined Contexts		
	New project	Corrective project	Non-corrective project
A	0.50	0.45	0.15
B	0.25	0.70	0.35
C	0.35	0.40	0.80
D	0.30	0.45	0.50
E	0.45	0.45	0.30

it does not have any missing task but chooses a different alternative: *Use cases definition*. On the other hand, the one tailored to context Corrective project exhibits a missing task: *System architecture definition*, but it neither adds any extra tasks nor chooses a different alternative.

7.5 Data Collection and Analysis

The former sections illustrated how different strategies yield different software processes for the *Requirements* activity. Here we report the comparison of the complete Mobius development process for the different contexts by measuring the number of extra or missing tasks of the template-based tailored process if compared to the optimal solution obtained by auto-

```
--Variability points: Requirements
helper def:rule1():Boolean=
  if(thisModule.getValue('Project type')='Non-corrective')
    then true
    else false
  endif;

--Optional Rules: Requirements
helper def:rule2():Boolean=
  if(thisModule.getValue('Project type') = 'New development' or
      thisModule.getValue('Interaction with other systems') = 'Complex')
    then true
    else false
  endif;

helper def:rule3():Boolean=
  if(thisModule.getValue('Usability')='High')
    then true
    else false
  endif;

helper def:rule4():Boolean=
  if((thisModule.getValue('Project type') = 'Non-corrective' or
      thisModule.getValue('Project type') = 'New development' ) and
      thisModule.getValue('Request') = 'No report')
    then true
    else false
  endif;

helper def:rule5():Boolean=if(thisModule.getValue('Request') = 'Report' or
    thisModule.getValue('Project type') = 'Corrective' )
  then true
  else false
endif;
```

Figure 7.10: Mobius automatic tailoring transformation.

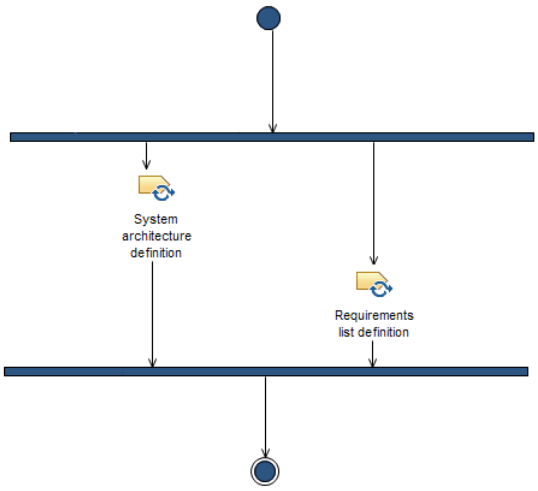
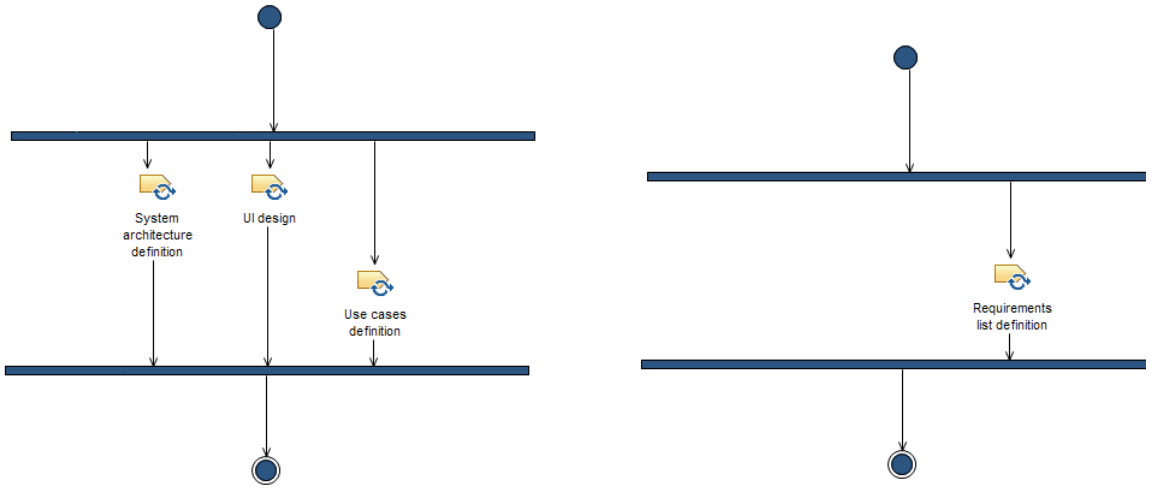


Figure 7.11: Mobius *Requirements* activity automatically tailored to Context E.

matic tailoring. In the first column of Tab. 7.17 we present the experimental contexts along with the closest predefined context according to Tab. 7.16. The second and third columns show the extra and the missing tasks in each case. The fourth column states the number of different tasks selected whenever an alternative variation point needs to be resolved. Notice that the last two lines correspond to context E; this is because both New project and Corrective project have exactly the same difference with E. However the results of selecting Corrective project are much better than those of selecting New project for all columns: extra tasks, missing tasks and different choices.

Similarly, we here report the comparison of the whole Rhiscom process for different contexts by measuring the number of extra or missing tasks of the template-based and automatic



(a) Mobius *Requirements* template-based tailored to context New project

(b) Mobius *Requirements* activity template-based tailored to context Corrective project

Figure 7.12: Predefined software process for experimental Context E.

Table 7.17: Comparison of the whole Mobius's process tasks in both strategies.

Process Comparison	**Difference in Tasks**		
	Extra tasks	**Missing tasks**	**Different alternative tasks chosen**
A - New project	3	3	3
B - Corrective project	10	8	0
C - Non-corrective project	3	2	3
D - Non-corrective project	3	8	3
E - New project	11	3	3
E - Corrective project	6	7	0

tailored processes. Different from Mobius's process whose only variability was optionality, Rhiscom's process also exhibits alternatives. The differences between strategies can be measured in extra or missing process elements, as well as different alternative choices whenever the variability is alternative. Table 7.18 summarizes these differences. The first column, presents the experimental contexts along with the closest predefined context. The second and third columns show the extra and the missing tasks in each case. The last column indicates the number of cases where a different alternative has been chosen. In this case both context B and D have two different predefined context equally distant.

We can see that for B, there is a big difference in productivity and correctness depending on the predefined context that is chosen. If we chose Corrective project, then the differences are much higher than those when choosing context Incident project. Similarly for D, if Incident project is chosen there is almost no difference, but if Corrective project is chosen instead, differences are much greater.

Table 7.18: Comparison of the whole Rhiscom process tasks in both strategies.

Process Comparison	Difference in Tasks		
	Extra tasks	Missing tasks	Different alternative tasks chosen
A - New project	0	0	0
B - Corrective project	3	6	2
B - Incident project	2	0	0
C - Corrective project	0	0	0
D - Corrective project	3	8	0
D - Incident project	0	0	2
E - Incident project	0	0	2

7.6 Results and Observations

We compared the suitability of use a template-based or an automatic MDE-based tailoring strategies in terms of productivity and correctness. The case study compared the software process tailoring in two Chilean software companies: Mobius and Rhiscom. The former includes a complex process with several variation points, while the latter is a simpler process with fewer variation points. For the automatic MDE-based tailoring, we computed the optimal process for each of the eight defined contexts. For the template-based strategy we considered the same eight contexts, three of them predefined as context-process pairs, and the remaining five as experimental contexts and for each of them we selected an approximate predefined process. Finally we compared the obtained process in each case using the EMF Compare tool in order to compare the resulting software processes.

In this study, we initially formulated a research question for evaluating the appropriateness of applying our integrated tool for tailoring software processes:

- **Considering productivity and correctness, what is the difference between the tailored software processes obtained using each strategy?**. The evidence suggest that there is always a difference between template-based and automatic MDE-based tailoring when the project context does not exactly coincides with the predefined context in the template-based strategy. This difference tends to be smaller for software process with little variability and larger for software process with several variation points. Automatic MDE-based tailoring encloses highly sophisticated tools, so construction costs are high if compared to template-based tailoring. The automatic tailoring presented has advantages over the predefined processes, even in the presence of limited variability points. These advantages are expressed as the complete precision of the output process for every context instance and therefore, it is adaptable for a company with increasing maturity. Even when the company has not evaluated to any level of CMMI, the results are guaranteed to be optimal. So far, the only theoretical advantage of the predefined processes is the ease of definition, because there is only a need for a few predefined processes, corresponding to the company's most common project types. As for Mobius and Rhiscom, we only defined three processes based on this assumption.

After analyzing both strategies in each company, we found that the automatic MDE-

based tailoring approach is almost always better, in terms of productivity and correctness, than the template-based tailoring approach. The only exceptions are the tailored processes for context A and C in the Rhiscom case, where both strategies obtained similar results. This confirms, using empirical data, something that is theoretically expected: the automatic tailoring strategy has more fine-grained rules for generating the process and the results of template-based tailoring could be sub-optimal. When comparing the differences in process productivity and correctness in each company, the impact of using an automatic or template-based tailoring strategy is more profound in the Mobius case. The numbers of extra and missing tasks are both greater in the company with most process variability points.

7.7 Lessons Learned

The only theoretical advantage of the predefined processes (template-based approach) is the ease of usage, since the company only needs to count on a process to support its most common project types. As for Mobius and Rhiscom, we only defined three processes based on this assumption.

For the predefined processes, if there are two context instances equally close to the new project's context, there is actually no preemptive way to decide which one is the best one. For the cases presented, there is the risk of not reaching the most suitable process when selecting a template process. This risk potentially reduces both the productivity and quality of the process, which is more harmful for small companies.

There is a need to count on a mechanism that offers a better solution when the distance from the target process context to two template-based process contexts is equal. A merge between the corresponding template-based processes would be a better "guess" for this case, even when they are similar. There is also a possibility for the predefined processes to be the initial mechanism for companies that want to include the initial levels of maturity in their software development.

On the other hand, automatic tailoring encloses highly sophisticated tools, where their construction costs are high if compared to template-based tailoring. However, the presented automatic tailoring (i.e., the use of these tools) has advantages over the use of predefined processes, even in the presence of limited variability points. These advantages are expressed as through the suitability (precision) of the output process according to every context instance. Therefore, this approach is adaptable for a company with increasing maturity. Even when the company was not evaluated to any level of CMMI, the results are usually good.

As with any empirical study, case study threats to validity should be identified and addressed in a systematic way. In our case, researchers are not assuming beforehand that one tailoring strategy is better than the other, but we found that our partners in software companies use a template-based software tailoring strategy. In this sense, we decided to compare the template-based and the automatic MDE-based tailoring, although there are probably other tailoring strategies used for such a purpose.

Regarding the external validity of results, a clear limitation of this study is that only two entities are used in the comparison. However, obtaining companies with formalized software process and using process-line approaches for coping with variation in not easy. Moreover, obtaining collaboration of the companies to open their process and applying new approaches is also a difficult endeavor. Similarities of the two companies are mainly in the previous formalization of their process, as well as being Chilean software companies. On the other hand, differences between these companies add diversity for comparing the tailoring strategies. The companies are different in key aspects such as application domains and lifetime in the market.

Other source of validation threats comes from the involvement of studies in the definition and tailoring of the software process-line. In this case study, researchers were actively collaborating with the software process practitioners for applying the tailoring strategies. To address this threat we used observer triangulation for collecting the data. Finally, the definition of variables to measure productivity and correctness of the adapted software processes was validated by people from the two Chilean software companies.

Chapter 8

ATAGeTT Validation

This chapter presents a complementary validation of the integrated tool for tailoring software processes in one additional software company. The validation was conducted using an explanatory case study where we evaluated expressiveness of the decision language and usability of the integrated tool.

Section 8.1 presents a motivation for conducting this explanatory case study. Section 8.2 shows the design of the case study that considers characteristics of ATAGeTT being evaluated, and also the stated research questions. Section 8.3 presents the setting of the case study and Section 8.4 illustrates its execution in a Chilean software company. Section 8.5 shows the results and data analysis. Section 8.6 discusses the results and Sec. 8.7 presents the threats to validity of the case study.

8.1 Motivation

In 2011, Adaptable Domain and Process Technology Engineering (ADAPTE)[1] project proposed by three Chilean universities (University of Chile, Technical University Federico Santa Maria, and Catholic University of Valparaiso) received funding from the Chilean government for a period of three years, with the aim of developing a tool that uses a MDE-based approach to tailor software process according to specific project contexts [65]. Four small Chilean software companies headquartered in Santiago, Chile participated in the ADAPTE project: Rhiscom, Imagen, Amisoft and Ki teknology.

Ki teknology formalized its development software process six years ago, as part of the ADAPTE project, and it applied MDE-based tailoring strategy to adjust its software processes, creating a Software Process Line (SPrL) [64]. To that end, the company specified three main components: (1) a general software process model that represents its organizational development software process (that includes variability), (2) a general context model that can be used to characterize particular projects, and (3) a tailoring transformation in

[1] ADAPTE web page: http://www.adapte.cl

ATL that includes tailoring rules for generating an adapted software process model. However, the use of the MDE-based tailoring strategy required expertise of the process engineer for defining models, writing and maintaining the tailoring transformation, and also mastering the transformation language syntax and semantics.

In order to support software process engineers during these activities we developed a Toolset for Automatically Generating Tailoring Transformations (ATAGeTT), which allows these engineers defining tailoring rules using the decision language (DL), automatically generating tailoring transformations using a HOT, and executing the software process tailoring [141]. The integrated tool (described in Chapter 6) hides the complexity of the automatic MDE-based tailoring and allows potential users to apply software process tailoring in a usable way. An explanatory case study was designed with the aim of gathering empirical evidence about the usability of integrated tool, expressiveness of the DL, and the feasibility of using it in the software industry.

8.2 Case Study Design

The explanatory case study followed the proposal of Yin [173]. Next we briefly introduce the integrated tool, describe the characteristics of ATAGeTT to be evaluated and present the research questions of the case study. Finally, we describe the case study.

8.2.1 Brief description of ATAGeTT

As part of the software process tailoring, we have to define tailoring rules as a model using the DL and automatically generated tailoring transformations in ATL using the presented HOT. In order to improve the usability and hide the complexity of MDE-based tailoring, we developed ATAGeTT [141, 144] for encapsulating and applying the automatic MDE-based software process tailoring for final users. This integrated tool balances the formality required by MDE and the usability needed by the users for software process tailoring. It also allows process engineers to interactively and transparently define tailoring rules, and automatically generate tailoring transformations. Moreover, the tool allows project managers to execute tailoring transformations for applying the proposed automatic MDE-based software process tailoring approach.

8.2.2 Description of characteristics to be evaluated

The aim of the explanatory case study is to evaluate the usability and expressiveness of ATAGeTT. Several authors, such as van Amstel et al. [158], Padda [112], Molina et al. [103] and Abrahao et al. [1], have defined some criteria for evaluating the usability of MDE concepts. There are also researchers like Felleisen [41], Paige et al. [113], Karsai et al. [79], Mohagheghi et al. [101], Barišić et al. [7], and Popovic et al. [116] that have defined some

Table 8.1: Validation factors of ATAGeTT.

Factors	Definition
Navigability	Whether users can move around in the application in an efficient way
Familiarity	Whether the user interface offers recognizable elements and interactions that can be understood by the user
Consistency	Degree of uniformity among elements of user interface and whether they offer meaningful metaphors to users
Flexibility	Whether the user interface of the software product can be tailored to suit users' personal preferences
Simplicity	Whether extraneous elements are eliminated from the user interface without significant information loss
Operability	Effort required to operate and control the software application

Table 8.2: Validation factors of DL.

Factors	Definition
Orthogonality	The language should be based on a few simple features, which can be combined to produce predictable results
Compatibility	The degree to which a DL is compatible with the development process
Scalability	It should be useful for modeling systems with a few components and interrelations, and systems with thousands of components and inter-relations
Suitability	The degree to which a DL is suitable to be used in the target domain

criteria validation for DSL. Considering these works, we defined the following evaluation criteria:

Usability is the level of learning, understanding, and memorizing of different concepts, relations, and also the knowledge about the when and why of using each concept. It usually implies counting on uniform notation, terminology, and features that are easy to learn, understand, and remember. In this sense, and considering the previously mentioned attributes, the usability of ATAGeTT was measured in terms of six factors: navigability, familiarity, consistency, flexibility, simplicity and operability. These factors are described in Table 8.1.

Expressiveness is measured by the ease and conciseness in which it is possible to express what is desired. On the other hand, the clarity of the language is a measure of how easy to understand and use it is for the users. This implies that the DL comes down to how clearly the language constructs can express the users' intentions. In this sense, the expressiveness of the DL is measured in terms of four factors: orthogonality, compatibility, scalability and suitability. These factors are described in Table 8.2.

8.2.3 Research questions definition

Taking into account the exploratory case study (presented in Chapter 7), which formulated a first research question, we conducted the explanatory case study considering the following two additional questions:

RQ2: How well does ATAGeTT work in practice?

Table 8.3: Characteristics of Ki teknology.

Business activity	**Age (years)**	**Certification**	**Type of offering**	**Clients**
Web development and design for the financial industry	20	ISO 9001:2008 (since 2004) CMMI Level 2 (since 2008)	Projects, products, services	>20

RQ3: How expressive is the decision language for specifying transformation rules to tailor software processes?

The RQ2 intends to understand the usefulness and usability of ATAGeTT in an industrial setting. This evaluation requires the participation of process engineers and project managers who evaluate the tool and provide their perception about the usability and usefulness of its graphical user interface.

The RQ3 intends to determine how suitable is the DL is for defining tailoring rules in terms of expressiveness. In order to evaluate the language expressiveness we have to verify that it counts on all the required constructs for expressing the process engineer's intentions for tailoring software process.

8.2.4 Description of the case study

In this explanatory case study participated only Ki teknology [2] personnel. This is a twenty years old Chilean company, located in Santiago city, that is certified in ISO 9001:2008 and CMMI Level 2. Ki teknology counts on an organizational software process based on Rational Unified Process (RUP), that is used to support developments of Web applications for the financial industry, particularly: projects (bespoke software development), products (COTS application development), and services (consulting, training, etc.).

In 2013, Ki teknology merged the best practices of RUP, CMMI level 2 and SCRUM in order to create the *Ki Agile Software Process*, which was designed by the process engineer and the SCRUM master of the company. Actually, the company has 95 clients and develops agile projects using such an agile process. A summary of the company characteristics is shown in Table 8.3.

According to the number of employees, Ki teknology is a medium-sized software company. It has sales of products and services in two other countries in addition to Chile. Table 8.4 shows the number of people that work in tasks specifically related to software development (e.g., project manager, developers, testers, analysts, etc.) and those who are involved in other tasks not related to software development (e.g., secretary, sales manager, cleaning staff, etc.).

[2]Ki teknology website: http://www.kiteknology.com

Table 8.4: Number of employees of Ki teknology.

Total (people)	People in software development tasks	People in other tasks
48	20 (42%)	28 (58%)

8.2.5 Units of analysis

The units of analysis represents the major entities being analyzed in the case study. In this case they were the end-users (i.e., the Ki teknology employees who evaluated the usability and usefulness of ATAGeTT) and ATAGeTT artifacts (e.g., the organizational software process, organizational context, and tailoring decisions). Next we briefly explain these units.

8.2.5.1 Ki teknology employees

There are two roles involved in automatic generation of transformations for software process tailoring in Ki teknology: the software process engineer and the project manager. The former is in charge of defining the organizational software process, the organizational context, and the tailoring rules. The latter is in charge of defining the project context (i.e., the characteristics of the project at hand) and generating the adapted software process.

8.2.5.2 Artifacts of ATAGeTT

There are two main artifacts involved in the automatic generation of transformations for software process tailoring: the decision language and the ATAGeTT software application. The first one allows the formal specification of tailoring rules, and it should be expressive enough as to grant generating rules in the software process domain. The second one is the tool that integrates several components, e.g., the organizational context definition, tailoring rules definition and project context definition. The components involve models and transformations that should be used in a transparent way, i.e., they should be invisible for the end-users.

8.3 Case Study Preparation

Table 8.5 shows the status of the artifacts of Ki teknology required to perform the automatic generation of transformations. The organizational software process with variability (i.e., the process that includes optional and alternative process elements) should be specified in EPF. The organizational context and tailoring decision that can be specified in other representations, should be reviewed and specified using ATAGeTT.

Table 8.5: Status of artifacts in Ki teknology.

	Organizational software process	**Organizational context**	**Tailoring decisions**
Current status (2015)	Defined and formalized in EPF (2013)	Defined and formalized in EMT (2013)	Defined in ATL (2013)
Future actions (2015-2016)	• Review the organizational software process. • Identify variable elements in the organizational software process. • Specify the variable elements in EPF.	• Review the organizational context. • Specify the organizational context in ATAGeTT.	• Review the tailoring decision for each variable element. • Specify the tailoring decisions in ATAGeTT.

8.4 Case Study Execution

In this explanatory case study, we used focus groups in each work session with the aim of understanding the usability and usefulness of ATAGeTT in a software industry scenario. From November 2015 to January 2016 various focus groups were conducted in Ki technology. Table 8.6 shows the dates, conducted activities, tools used in the sessions, number of work sessions, total of hours and teams participating in the case study.

This case study considered four main activities: (1) reviewing of the organizational software process of Ki teknology for updating software process elements and their variable elements, (2) reviewing of the organizational context for updating context attributes and their values, (3) reviewing of the tailoring rules for identifying variable software process elements (and labeling them as annotations in the organizational software process), and (4) using ATAGeTT in practice for tailoring the Ki software process. The EPF was used to support this experience, since the participants were familiar with such a tool. Therefore, ATAGeTT was used for applying the automatic MDE-based software process tailoring and the EPF to visualize the result.

The focus groups lasted eleven hours in five work sessions; the first work session lasted 3 hours, and 2 hours for the others. Each session was conducted in the software company's offices involving the company and research teams. A total of four Ki employees participated during the focus group sessions: one person from the chief commercial office, two people from the process and product quality assurance area, one person from the software development team. Moreover, one person from the research team (the author of this thesis). Next we present the activities conducted on them.

Table 8.6: Summary of work sessions.

Date	**Work sessions**	**Hours (total)**	Company team	**Research team**	**Activities**	**Tools**
November 2015 - January 2016	5	11	4	1	• Review the organizational software process • Review the organizational context • Review the tailoring decisions • Apply ATAGeTT for tailoring the Ki software process	EPF and ATAGeTT

Table 8.7: Summary of revisions of Ki teknology's organizational software process.

Version	**Software process based on**	**Tasks**	**Roles**	**Work products**	**Variable elements**
2013	SCRUM and RUP	30	16	17	0
2016	SCRUM and RUP	41	16	20	17 optional elements

8.4.1 Review of the organizational software process

Ki agile software process was formalized in 2013 without variability. In order to validate this software process, the research and the company team reviewed such a process (the descriptive software process) in three work sessions. The software process included information about the tasks, roles and work products. It also included information about the optional variability in the software process. Figure 8.1 shows a snapshot of the Ki agile software process represented in EPF, and highlights *Transfer*, a task as an optional element (marked in the last column). Figure 8.2 shows a snapshot of this software process in its Web version, which was generated automatically using EPF.

Table 8.7 shows a summary of software process that was reviewed; i.e., the Ki agile software process (2016), which had 11 new process elements and 17 new variable (optional) elements. In this case, Ki agile software process does not have alternative elements.

8.4.2 Review of the organizational context

The organizational context of Ki teknology was specified in 2010 and reported in [65]. However, this context model was used to characterize the project contexts for tailoring requirements engineering process that was part of organizational software process based on RUP. In the fourth work session, we reviewed this organizational context model (Table 8.8) that has five attributes: *Customer type* that can assume two potential values: *New* and *Old*; *Project type* that can have three potential values: *UX*, *Development* and *Design and devel-*

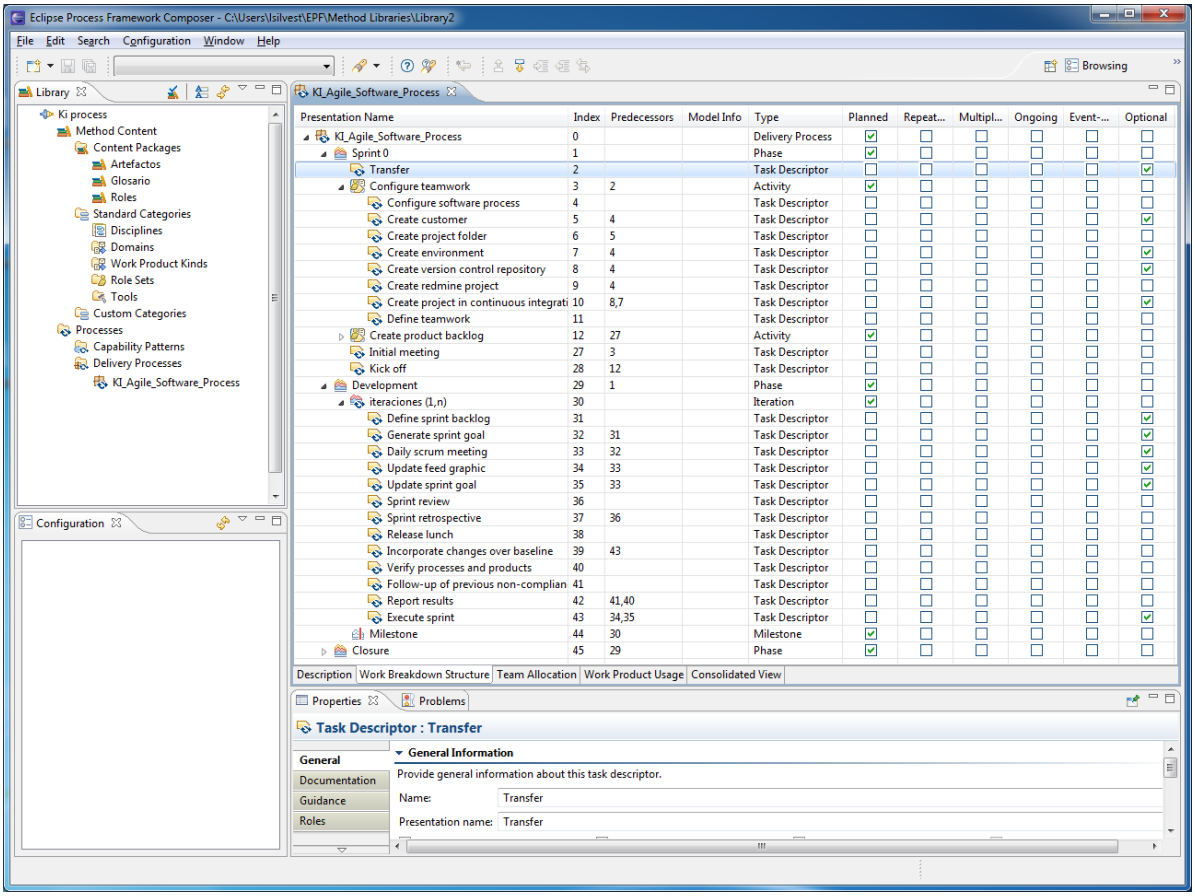


Figure 8.1: Software process of Ki teknology (EPF version).

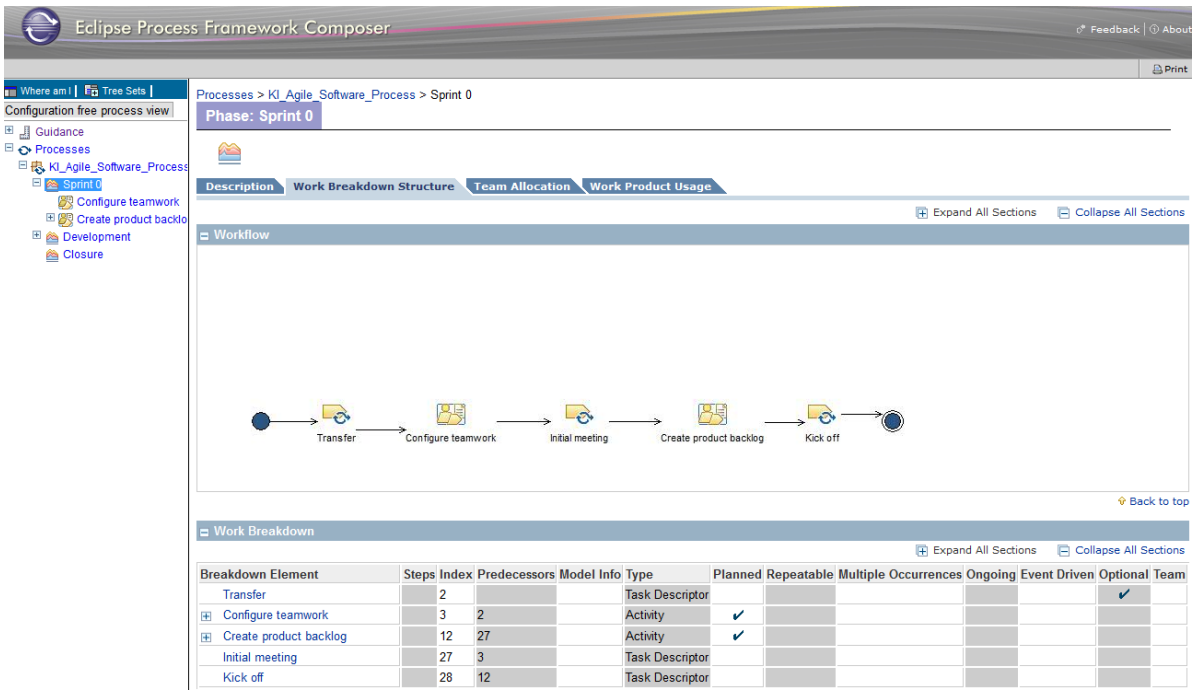


Figure 8.2: Software process of Ki teknology (Web version).

Table 8.8: Organizational context of Ki teknology

Dimensions	Attributes	Values
Process	Customer type	New Old
	Project type	UX Development Design and development
	Sales source	Customers service Other divisions
	Customer with maintenance	Yes No
	Factory adopts Ki process	Yes No

opment; *Sales source* that can be: *Customers service*, and *Other divisions*; *Customer with maintenance* that can be: *Yes* and *No*; and *Factory adopts KI process* that can assume two potential values: *Yes* and *No*. These context attributes are grouped in one dimension for better comprehension: *Process*.

8.4.3 Review of the tailoring decisions

The tailoring decisions of Ki teknology were also specified in 2010 for tailoring purposes [65]. These decisions were specified in ATL and only considered the tailoring of the requirements engineering process. In the fourth work session, we reviewed these tailoring decisions, and put notes in the organizational software process for each variable element involved in these decisions. Figure 8.3 shows a snapshot of Ki agile software process in EPF that includes notes in the variable elements. These notes describe conditions and conclusions as text. We can also observe red notes for describing optional elements and green notes for alternative elements. Figure 8.3 shows red notes to indicate tailoring decisions in several tasks, e.g., in *Create customers*, *Create version control repository*, *Create environment* and *Create project in continuous integration tool*.

We identified seventeen variable process elements with its notes that include tailoring decisions (see Table 8.9). Each tailoring decision was described through the following information (table columns): process element, variability type, conditions and conclusions.

Table 8.9: Tailoring Decisions of Ki teknology

Decision	Process Element	Variability Type	Conditions	Conclusion
Decision 1	Transfer	Optional	(Sales source=Other divisions)	Remove: Transfer
Decision 2	Create customer	Optional	(Customer type=Old)	Remove: Create customer
Decision 3	Create environment	Optional	(Customer with maintenance=No) OR (Factory adopts KI process=No)	Remove: Create environment

Decision 4	Create version control repository	Optional	(Customer with maintenance=No)	Remove: Create version control repository
Decision 5	Create project in continuous integration tool	Optional	(Project type=UX)	Remove: Create project in continuous integration tool
Decision 6	Estimate user story	Optional	(Factory adopts KI process=No)	Remove: Estimate user story
Decision 7	User experience	Optional	(Project type=Development) OR (Factory adopts KI process=No)	Remove: User experience
Decision 8	Benchmarking	Optional	(Project type=Development) OR (Factory adopts KI process=No)	Remove: Benchmarking
Decision 9	Create interaction scores	Optional	(Project type=Development) OR (Factory adopts KI process=No)	Remove: Create interaction scores
Decision 10	Define team speed	Optional	(Factory adopts KI process=No)	Remove: Define team speed
Decision 11	Create burn up	Optional	(Factory adopts KI process=No)	Remove: Create burn up
Decision 12	Define sprint backlog	Optional	(Factory adopts KI process=No)	Remove: Define sprint backlog
Decision 13	Generate sprint goal	Optional	(Factory adopts KI process=No)	Remove: Generate sprint goal
Decision 14	Daily scrum meeting	Optional	(Factory adopts KI process=No)	Remove: Daily scrum meeting
Decision 15	Update feed graphic	Optional	(Factory adopts KI process=No)	Remove: Update feed graphic
Decision 16	Update sprint goal	Optional	(Factory adopts KI process=No)	Remove: Update sprint goal
Decision 17	Execute sprint	Optional	(Factory adopts KI process=No)	Remove: Execute sprint

8.4.4 Apply ATAGeTT for software process tailoring

In the fifth work session we applied ATAGeTT for defining the organizational context, defining tailoring rules, generating tailoring transformations, defining project contexts, executing software process tailoring and visualizing adapted software process. In the next subsections, we presented the application of ATAGeTT for adapting Ki agile software process to a concrete

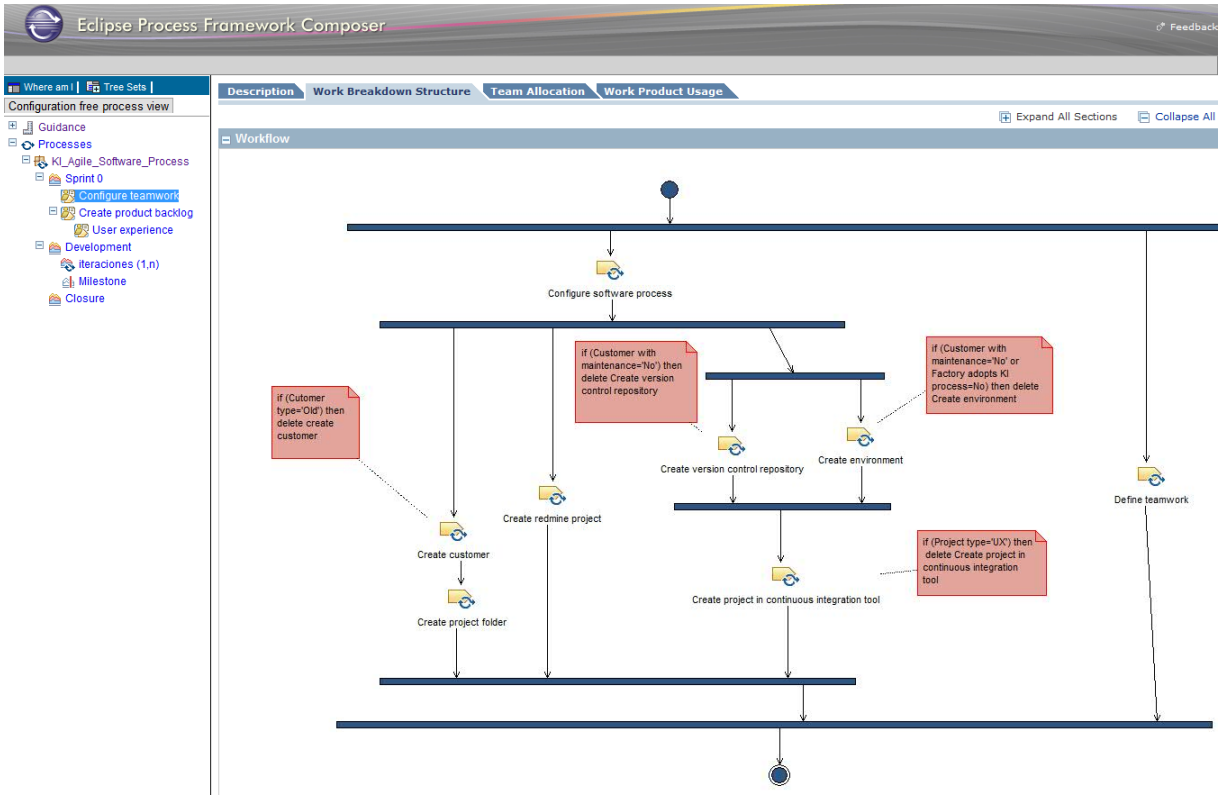


Figure 8.3: Tailoring decisions of Ki teknology's software process

project context.

8.4.4.1 Defining the organizational context

In Ki teknology the software process engineer is in charge of defining the organizational context, which includes dimensions, context attributes and context attributes values (Table 8.8). Figure 8.4 shows the organizational context of Ki teknology in the user interface of ATAGeTT. In this case, the organizational context was called *KIOrgContext*.

The software process engineer can edit the organizational context by adding, removing or editing dimensions, attributes and attribute values. Moreover, this engineer can also save the organizational context, as XML file, using the *create context model* button. The organizational context is stored in a predetermined folder of the ATAGeTT environment.

8.4.4.2 Defining the tailoring rules

The software process engineer is also in charge of defining the tailoring rules. These rules were specified in Table 8.9, and they include process elements, variability type, conditions and conclusions. To that end, the definition of rules requires information about the organizational software process and the organizational context.

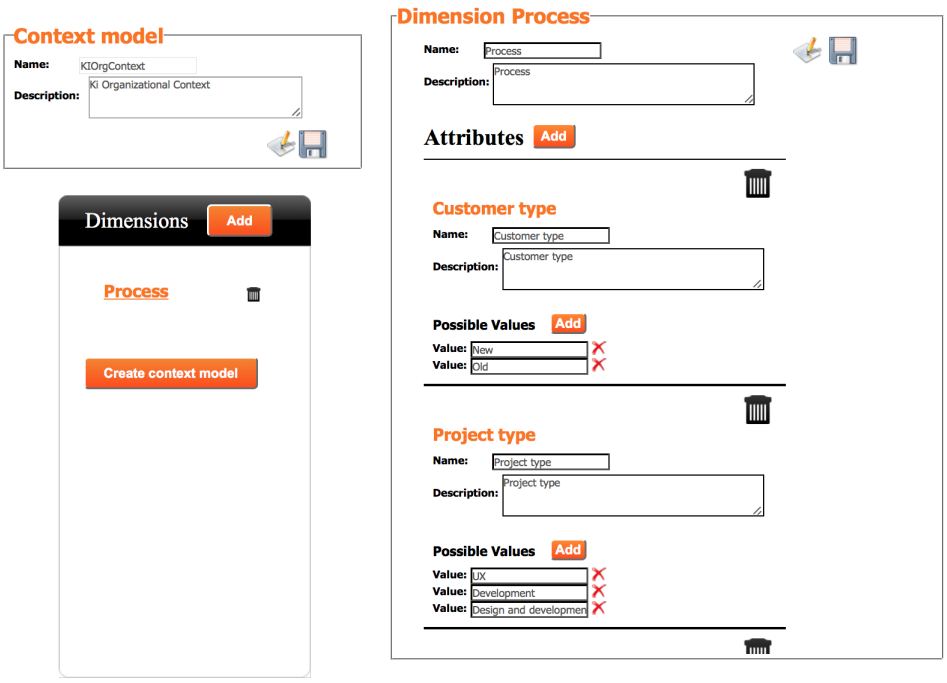


Figure 8.4: Ki teknology's organizational context.

Figure 8.5 shows the selection of organizational software process and organizational context using ATAGeTT. The organizational software process was called *KIOrgProcess* and it was exported from EPF as a XML file. The organizational context was called *KIOrgContext* and it was generated from the definition of organizational context, also as a XML file. Moreover, the organizational software process is transformed in the organizational software process model using the proposed *Injector*.

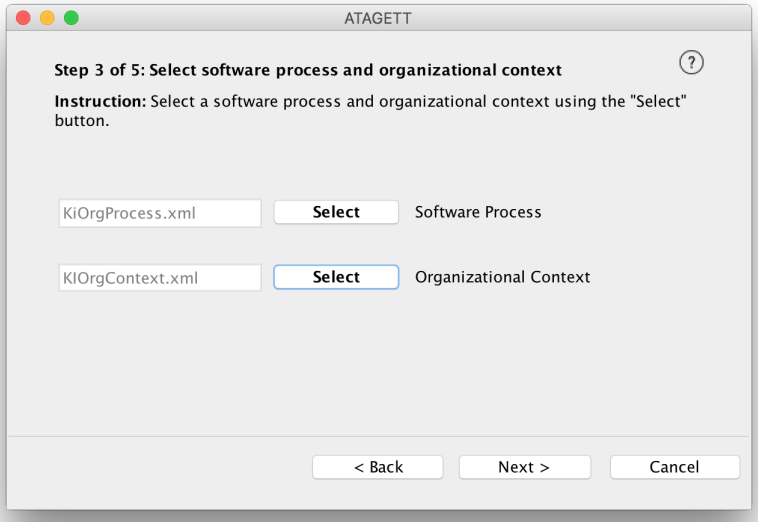


Figure 8.5: Selecting organizational software process and organizational context.

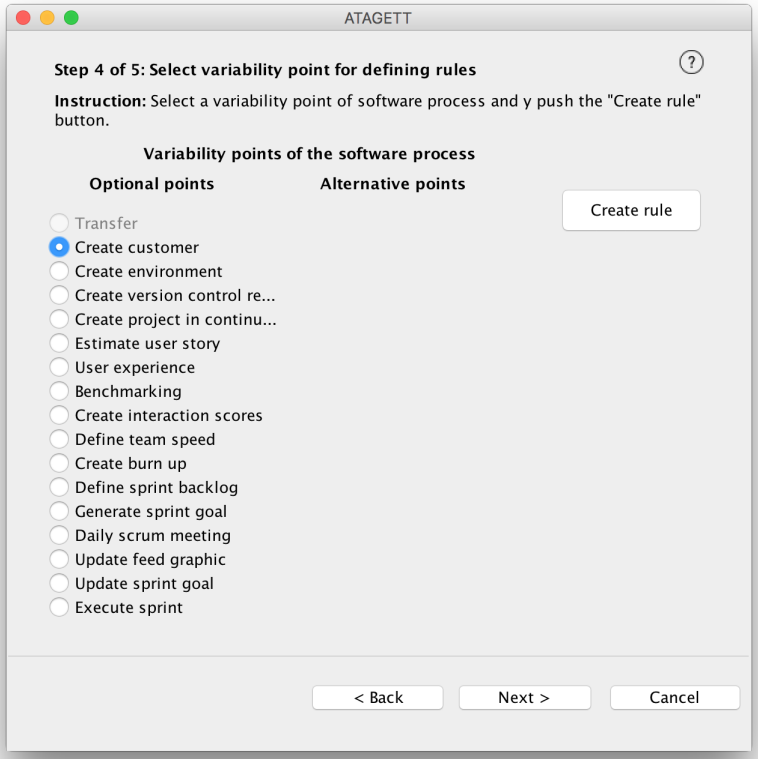


Figure 8.6: Selecting variability points for defining tailoring rules.

Once the organizational context and the organizational software process have been selected, ATAGeTT allows the software process engineer to select a variability point of the software process for defining tailoring rules. Figure 8.6 shows a list of variable process elements that can be obtained from *KIOrgProcess*. The process engineer can select a variable process element and define tailoring rules. For example, he/she can select *Create customer* and click on *Create rule* button.

Figure 8.7 shows the definition of tailoring rules using our DSL and ATAGeTT. The definition of tailoring rules considers the information of organizational context, i.e., the context attributes and their values. The software process engineer can define conditions and conclusions for each variable process element using the graphical environment. For example, we can define a tailoring rules to *Create customer* with a condition: *Customer type* is *Old*, and conclusion: *Remove Create customer*.

After defining tailoring rules, ATAGeTT generates the variation decision model of Ki teknology. Figure 8.8 shows this model as XMI file, which can be visualized in EMF. For example, we can identify a *Rule* with conditions and conclusion for *Create customer* in *Configuration Rule* section. The variation decision model is stored in a predetermined folder of the ATAGeTT environment.

Finally, ATAGeTT allows the software process engineer to execute the HOT and the ATL

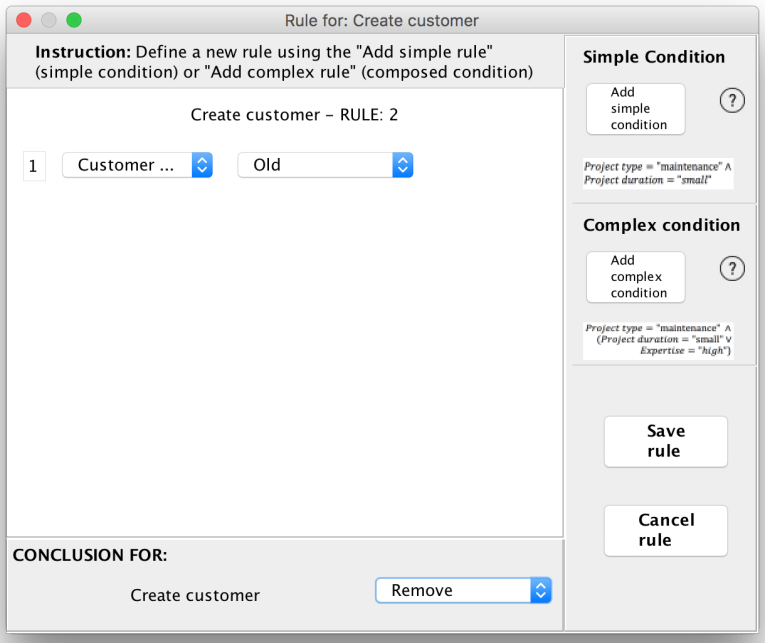


Figure 8.7: Defining tailoring rules.

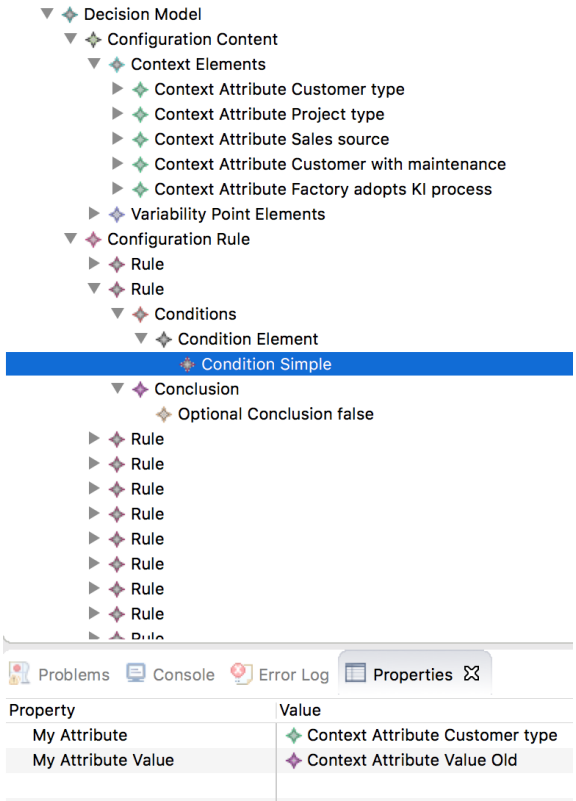


Figure 8.8: Variation decision model of Ki teknology as XMI file.

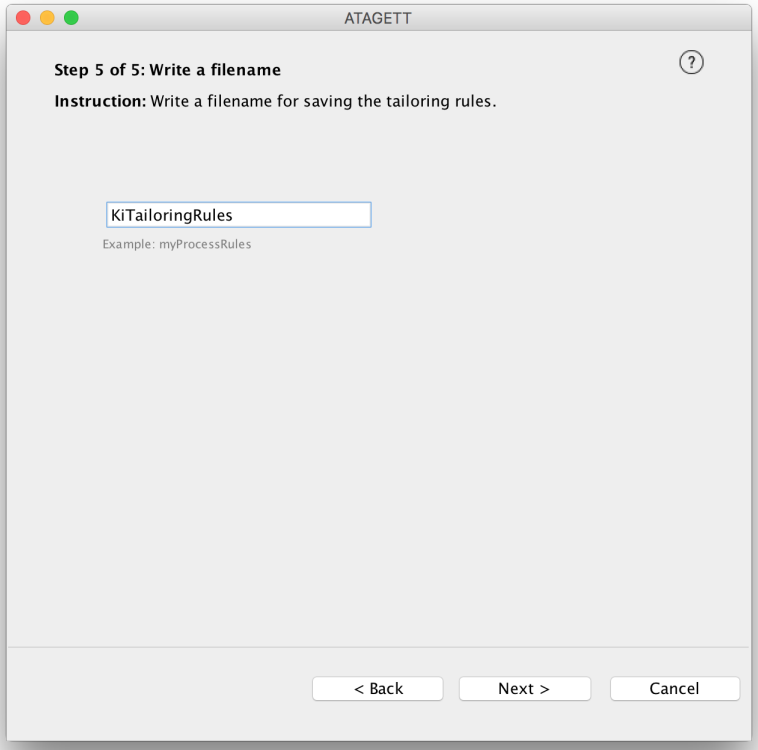


Figure 8.9: Generating tailoring transformation of Ki teknology.

extractor for automatically generating the tailoring transformation. To that end, the graphical environment automatically uses the variation decision model generated in the definition of tailoring rules.

The process engineer defines a name for the transformation, and then the graphical environment executes the HOT and the ATL extractor in automatic way. Figure 8.9 shows the graphical environment where the software process engineer writes a name of the tailoring transformation. In this case, we can write *KiTailoringRules* as such a name. The tailoring transformation is generated automatically when the user clicks on the *Next* button.

Figure 8.10 shows the tailoring transformation (as ATL code) that can be visualized in EMF. We can see that the tailoring rule for *Create customer* is called *ruleOpt2* (see line 155 in Fig. 8.10). This transformation is stored in a predetermined folder of ATAGeTT environment.

8.4.4.3 Defining the project context

As mentioned, ATAGeTT allows a project manager to define and execute software process tailoring in an automatic way. In order to do that, the project manager defines a project context by setting the values of the organizational context for the project at hand. Figure 8.11

```
148 helper def: ruleOpt1(): Boolean =
149     if (thisModule.getValue('Sales source') = 'Other divisions') then
150         false
151     else
152         true
153     endif;
154
155 helper def: ruleOpt2(): Boolean =
156     if (thisModule.getValue('Customer type') = 'Old') then
157         false
158     else
159         true
160     endif;
161
162 helper def: ruleOpt3(): Boolean =
163     if (thisModule.getValue('Customer with maintenance') = 'No' or thisModule.
164             getValue('Factory adopts KI process') = 'No') then
165         false
166     else
167         true
168     endif;
169
170 helper def: ruleOpt4(): Boolean =
171     if (thisModule.getValue('Customer with maintenance') = 'No') then
172         false
173     else
174         true
175     endif;
176
177 helper def: ruleOpt5(): Boolean =
178     if (thisModule.getValue('Project type') = 'UX') then
179         false
180     else
181         true
182     endif;
```

Figure 8.10: Tailoring transformation of Ki teknology as ATL code.

shows a graphical environment for defining a concrete context, called *TurBus New Project*. In this case, the concrete context is a new project where *Customer type* is *New*, *Project type* is *Development*, *Sales source* is *Other divisions*, *Customer with maintenance* is *No* and *Factory adopts KI* process is *Yes*. The project manager can then generate the concrete context as model by clicking on the *Configure concrete context* button.

Once defined the project context, ATAGeTT executes the automatic MDE-based software process tailoring by taking the project context model and the organizational software process model as input, and generating (using the tailoring transformation) the adapted software process model as output. Figure 8.12 shows the graphical environment for executing software process tailoring, which encapsulates the MDE-based software process tailoring using the ATL launcher. This launcher is a standalone application (developed by the author of this thesis) that implements model transformations in programmatic way. In this case, the project manager can execute the ATL launcher by clicking on the *Execute software process tailoring* button. Finally, the project context model and the adapted software process model are in a predetermined folder of the ATAGeTT environment.

8.4.4.4 Visualizing the adapted software process

The project manager requires visualizing the adapted software process in EPF. In order to do that, ATAGeTT takes the adapted software process model and generates the adapted software process as XML file using the *Extractor* presented in Chapter 3.

Figure 8.13 shows the adapted software process that can be visualized in EPF. There we can see that the software process of Ki teknology includes, in this case, all the process

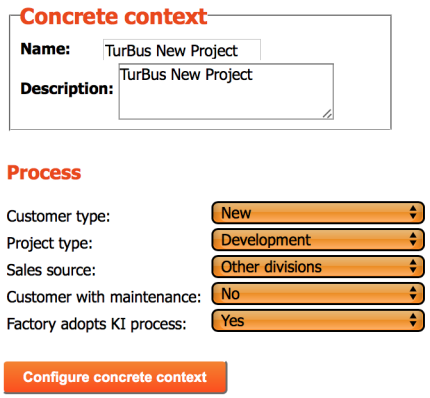


Figure 8.11: Defining a concrete context of Ki teknology.

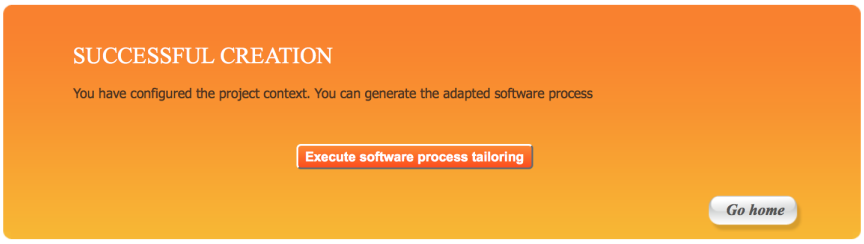


Figure 8.12: Executing Ki teknology's organizational context.

elements because the *TurBus New Project* requires all activities, tasks, artifacts and roles.

8.5 Data Collection and Analysis

In the former four sections, we applied the software process tailoring using ATAGeTT, and in the fifth session we collected and stored multiple sources of evidence for evaluating ATAGeTT. We therefore defined three sources of evidence: surveys, observations and records. Finally, we analyzed the sources of evidence for evaluating the usability and expressiveness of ATAGeTT.

8.5.1 Data collection

In the data collection, we considered the result of defining the organizational context, tailoring rules and project context. These activities were described and applied in the section 8.4.4 using particular tools that are part of ATAGeTT. In the following subsections, we describe the data gathering activities and the artifacts resulting from them.

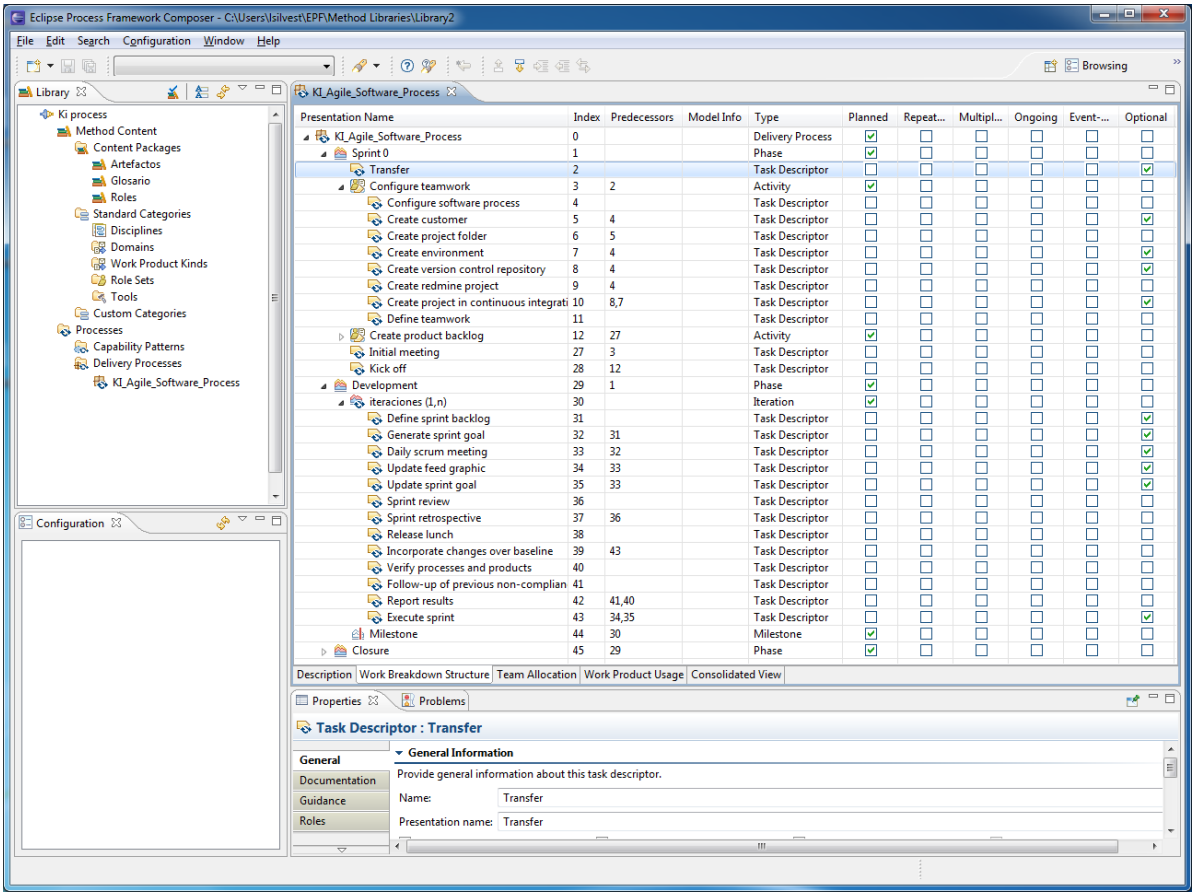


Figure 8.13: Software process of Ki teknology (EPF version)

8.5.1.1 Description of data gathering

We collected quantitative and qualitative data during the fifth work session with the Ki teknology team. Table 8.10 shows the main characteristics of data gathering session with the support of two process engineers and one project manager of the company. The former are in charge of defining organizational context and defining tailoring rules that allow to build the tailoring transformations. On the other hand, the latter is in charge of defining project contexts that determine how to execute the software process tailoring.

As mentioned before, at the end of each activity we consider three sources of evidence:

Table 8.10: Characteristics of data gathering.

Ki team	**Activities**	**Source of evidence**	**Evaluation artifacts**
2 Software process engineers	– Defining organizational context – Defining tailoring rules	Survey Observation Record	Form Logbook Audio and video
1 Project manager	– Defining project context	Survey Observation Record	Form Logbook Audio and video

surveys, observations and records. In order to do that, we defined artifacts for each source of evidence: forms (survey), logbook (observation), and audio/video record (record). Next we briefly explain each of them.

Surveys consider a series of questions designed to gather quantitative and qualitative information from the participants. At the end of each activity, we deployed particular surveys for evaluating usability and expressiveness of ATAGeTT (see Annex B).

Observations are watching activities carefully conducted by someone or something (e.g., an agent) for gathering quantitative and qualitative data from or about the units of study. During each activity, we used a logbook for obtaining complementary data from the end-users.

Records are documents that memorializes and provides objective evidence about the activities performed by the participants. During each activity, the author of this thesis requested a written authorization for recording (in audio and video) the activities the users perform on the computer desktop while using ATAGeTT.

8.5.2 Description of evaluation artifacts

In the fifth work session we also considered three surveys and three video records for each activity of data gathering, and one audio record of the complete session. These artifacts were used to establish a triangulation of the empirical data, which involved the use of more than one data collection method on the same topic. In the following subsections, we describe the evaluation artifacts.

8.5.2.1 Evaluation forms

A particular form was designed for evaluating each activity; i.e., the organizational context, tailoring rules and project contexts. Each evaluation form had two sections: a survey and a questionnaire.

The survey section considered a set of sentences for validating each evaluation factor of ATAGeTT and DL, which were described in table 8.1 and table 8.2 respectively. The evaluation score related to these sentences was specified in a five-point Likert scale. On the other hand, the questionnaire section considered two questions that try to capture the users feeling about the usefulness and adoption effort of ATAGeTT, and it also asked for the points of improvement of the tool. Next, we present the description of the three surveys.

Organizational context survey. The survey form had 33 sentences and two questions. All of them focused on evaluating quality attributes related to the organizational context definition tool. The sentences were organized in eight sections: six of them for evaluating usability factors of the tool, one section for evaluating the software operability, and one section for evaluating other quality factors. The questions section considered the following two questions: the first one (closed-ended question) focused on identifying the usefulness of the tool for defining the organizational context, and the second one (open-ended) asked

about improvement aspects of the tool. The organizational context survey is presented in Annex B, Section B.1.

Tailoring rules survey. This survey form had 49 sentences and two questions. All of them focused on evaluating the usability and usefulness (expressiveness) of the tailoring rules definition tool. The sentences were organized in eleven sections: nine for evaluating usability factors, one for evaluating expressiveness of the decision language, and one section for evaluating other software quality factors of the user interface that allows the user to define the tailoring rules. Similar to the previous case, the questions section considered two questions: one closed-ended and the other open-ended. The first one intended to determine adoption effort of the tool and the second one asked for improvement aspects of such an ATAGeTT component. The tailoring rules survey is presented in Annex B, Section B.2.

Project context survey. This survey had 32 sentences and two open questions that evaluated the project context definition tool. The sentences were organized in eight sections for evaluating usability factors, one section for evaluating operability factors, and one section for evaluating software quality factors of the project context definition tool. Once again, the final questions were two: one closed-ended and the other open-ended. The first one focused on determining the usefulness of the tool, and the second one focused on the improvement aspects of such a service. The organizational context survey is presented in Annex B, Section B.3.

8.5.2.2 Logbook of observations

The author of this thesis used a logbook to obtain complementary information about the usability and expressiveness of ATAGeTT. The logbook had three sections for each activity of data gathering. Each section considered information about usability, expressiveness, timing, usefulness and improvement aspects of ATAGeTT.

8.5.2.3 Record of audio and video

During the fifth work session we recorded audio and video of the users while they performed the activities in their desktop applications, i.e., during the evaluation in practice of ATAGeTT. In this case we also obtained a written authorization from the users for recording audio and video of their activities. These records considered complementary information about usability, expressiveness, timing, usefulness, adoption effort and possible improvements of ATAGeTT.

8.5.3 Data analysis

In the data analysis, we considered three potential users: two process engineers and one project manager. We also considered several sources of evidence for each activity: five surveys, three videos of the user activities, one audio record, and one logbook. These resources have information about activity timing (records and logbook), usability and expressiveness

Table 8.11: Activity timing for adapting the Ki agile software process to a concrete project context.

	Activity	**Activity elements**	**Activity timing**	**Total timing**
Software process engineer 1	Defining organizational context	6 context attributes	3.18 min.	16.58 min.
	Defining tailoring rules	17 tailoring rules	13.40 min.	
Software process engineer 2	Defining organizational context	6 context attributes	4.32 min.	18.42 min.
	Defining tailoring rules	17 tailoring rules	14.10 min.	
Project manager	Defining project context	1 context configuration	1.35 min.	1.35 min.

factors (surveys and logbook), usefulness, adoption effort, and improvement aspects (surveys and logbook). We performed a quantitative analysis of this information. Next, we describe each variable and the results obtained from these resource.

8.5.3.1 About activity timing for adapting the Ki agile software process

Table 8.11 shows the activity timing for defining the organizational context, defining tailoring rules (including tailoring transformations generation) and defining project contexts (including software process tailoring execution). Notice that the organizational context considers six context attributes, the tailoring rules considers seventeen rules and the project context considers one configuration for a particular project.

The "software process engineer 1" performed the organizational context definition in 3.18 minutes and the tailoring rules definition in 13.40 minutes; therefore, both activities required 16.58 minutes in being completed. Moreover, the "software process engineer 2" performed the organizational context definition in 4.32 minutes and the tailoring rules definition in 14.10 minutes; therefore, both activities required 18.42 minutes. In this case, both process engineers had the same professional qualifications in terms of technical skills and job experience in software process analysis, therefore, their performances are comparable. In consequence, the average for defining the organizational context and the tailoring rules was 17.50 minutes. After that, the project manager performed the project context definition in 1.35 minutes. Although we can argue that the system is responsible for performing this activity in a very short time period, we have also to recognize this last user had extensive experience planning and managing software process.

Although these results are promising, it is not possible to guarantee the same performance for other software companies, because the number of context attributes and tailoring rules can increase or decrease the time spent in these model definition activities. However, we consider highly positive the fact that software process engineers do not have to interact with models, transformations, and ATL code for tailoring the software process.

Table 8.12: Evaluation of timing for each activity that is used for the adapting Ki agile software process.

	Evaluation	**Number of items**	**Evaluation timing**	**Total timing**
Software process engineer 1	Organizational context survey	33 sentences 2 questions	12.10 min.	27.30 min.
	Tailoring rules survey	49 sentences 2 questions	15.20 min.	
Software process engineer 2	Organizational context survey	33 sentences 2 questions	13.50 min.	30.28 min.
	Tailoring rules survey	49 sentences 2 questions	16.38 min.	
Project manager 1	Project context survey	32 sentences 2 questions	12.42 min.	12.42 min.

8.5.3.2 About usability and expressiveness of ATAGeTT

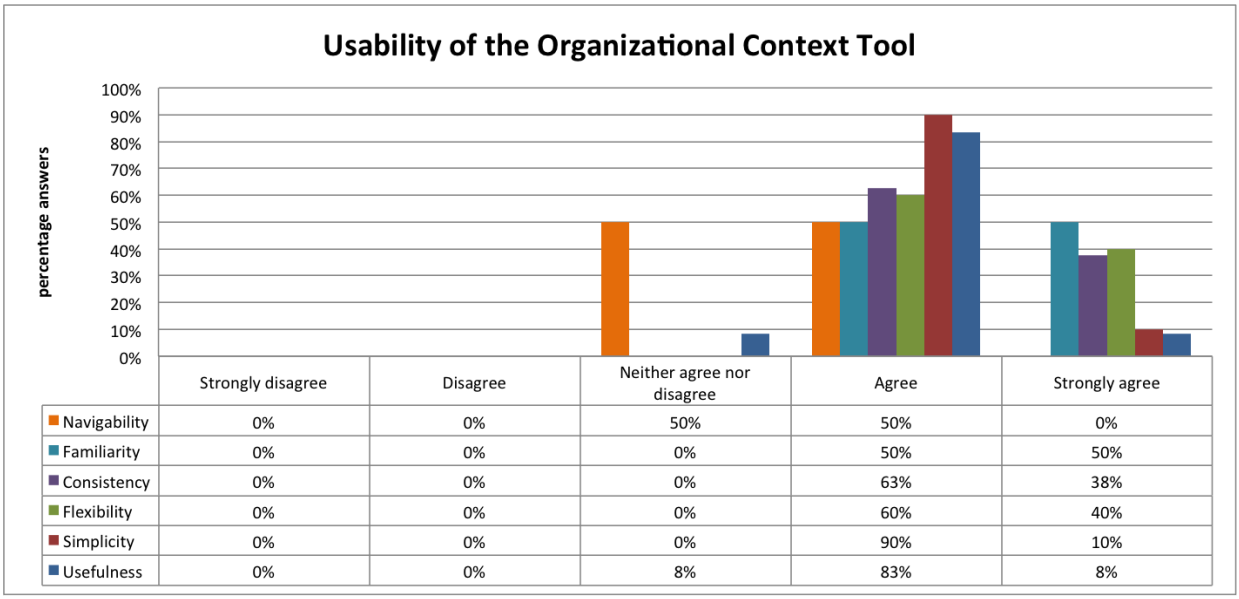
Anonymous surveys were used with the software process engineers and the project manager to evaluate the usability and expressiveness of ATAGeTT. Each evaluation was represented through one or more sentences (items) that considered particular factors: usability, operability and other quality factors of the tool, and also the expressiveness of the decision language. The sentences are evaluated using a 5-point Likert scale (strongly disagree, disagree, neither agree nor disagree, agree, strongly agree). In the following subsections, we analyze the results of each evaluation.

Evaluation of the organizational context definition

In this case, we evaluated the usability, operability and other software quality factors of the ATAGeTT component that allows the user to define the organizational context of the company. These evaluations were performed by two software process engineers; next, we explain them.

Usability. The usability was evaluated in terms of navigability (2 sentences), familiarity (3 sentences), consistency (4 sentences), flexibility (5 sentences), simplicity (5 sentences), and usefulness (6 sentences). Figure 8.14 presents the results of the usability evaluation. The software process engineers considered that the organizational context definition tool showed good *usability* in terms of familiarity (50% agree and 50% strongly agree), consistency (62% agree and 38% strongly agree), flexibility (60% agree and 40% strongly agree), simplicity (90% agree and 10% strongly agree) and usefulness (83% agree and 8% strongly agree). However, the software process engineers were neutral about the navigability of the organizational context definition tool.

The survey results shows a tendency towards agree or strongly agree about the usability of the tool, and it offers a good user experience in the organizational context definition. Moreover, we can indicate that the context definition tool was usable in practice because the 83% of usability factors were evaluated between *agree* and *strongly agree*. However, we



	Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree
Navigability	0%	0%	50%	50%	0%
Familiarity	0%	0%	0%	50%	50%
Consistency	0%	0%	0%	63%	38%
Flexibility	0%	0%	0%	60%	40%
Simplicity	0%	0%	0%	90%	10%
Usefulness	0%	0%	8%	83%	8%

Figure 8.14: Distribution of the usability results according to the Likert scale.

recognize that the tool requires improvements in terms of navigability.

In order to understand the navigability results, we analyzed the logbook to obtain complementary information. In this sense, we obtained suggestions to improve the navigability of the tool. Next, we present some quotes with these suggestions:

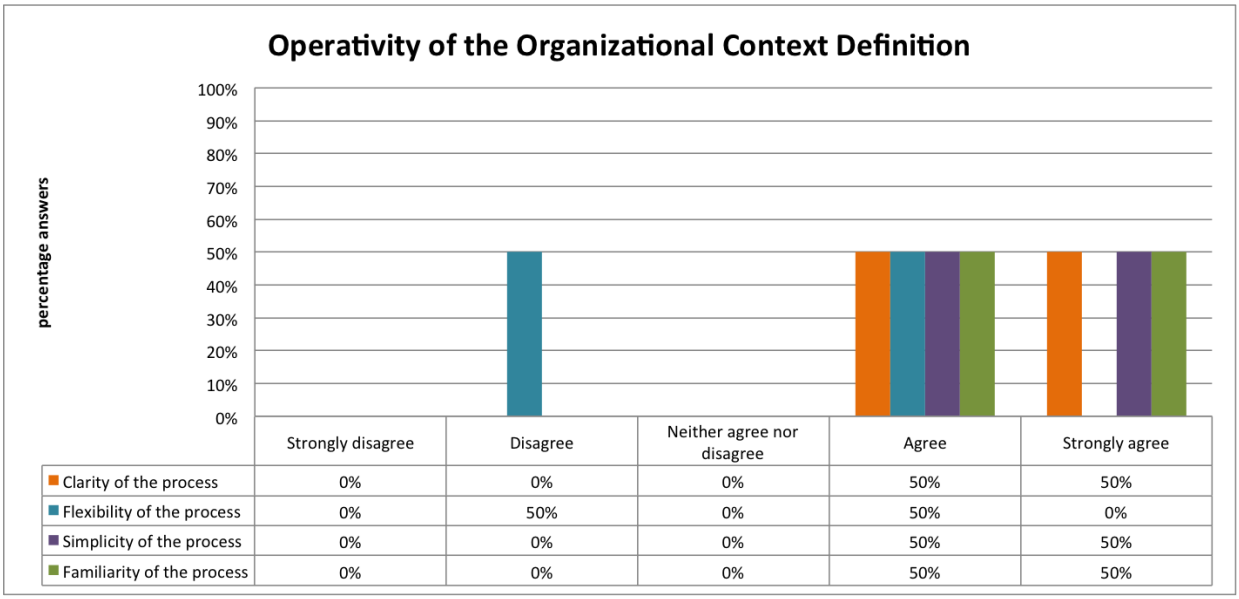
"The use of traceability links can facilitate the navigation between the user interface and the process elements".

"Knowing the current location of the user in the interface of the system can ease the definition of the organizational context process".

Operability. This attribute was evaluated in terms of clarity (1 sentence), flexibility (1 sentence), simplicity (1 sentence) and familiarity (1 sentence) of the organizational context definition tool. Figure 8.15 presents the obtained results. The software process engineers were positive (*agree* and *strongly agree*) about the operability of the tool, in terms of its clarity (50% agree and 50% strongly agree), simplicity (50% agree and 50% strongly agree), familiarity (50% agree and 50% strongly agree). However, the software process engineers were partially negative about the flexibility the organizational context definition provides to address the process tailoring needs (50% disagree and 50% agree).

The survey results suggest that the organizational context definition is operable because it minimizes the time and effort needed by the software process engineers. In this sense, 75% of operability factors were evaluated positively (with *agree* and *strongly agree*). Moreover, the evidence indicates the organizational context definition worked well in practice and it was addressable for the end-users. However, we can recognize the tool requires improvements in terms of flexibility.

In order to understand the flexibility results, once again we analyzed the logbook for



	Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree
Clarity of the process	0%	0%	0%	50%	50%
Flexibility of the process	0%	50%	0%	50%	0%
Simplicity of the process	0%	0%	0%	50%	50%
Familiarity of the process	0%	0%	0%	50%	50%

Figure 8.15: Distribution of the operability results according to the Likert scale.

obtaining complementary information. In this sense, we obtained comments about flexibility limitations that can be attributed to either: (1) *accessibility of unusual actions* and (2) *personalization of the interface elements*. Next, we present some quotes from the comments made by the software process engineers.

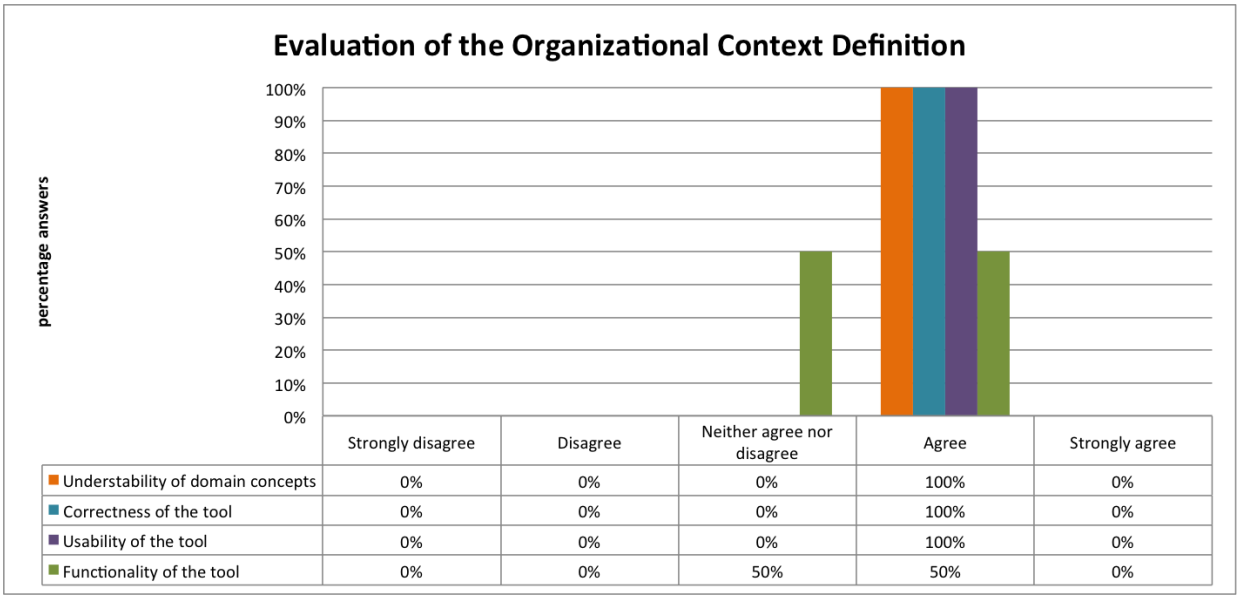
"The tool does not have enough flexibility for considering unpredicted actions of the user during the context attributes definition".

"Probably, other kind of customization of the system can facilitate the organizational context definition".

Software quality factors. The software quality factors were evaluated in terms of understandability, correctness, usability and functionality of the tool (4 sentences). Figure 8.16 presents the obtained results. The software process engineers were positive (*agree*) about the quality of the organizational context definition tool, in terms of the understandability of the domain concepts (100% agree), correctness (100% agree) and usability of the tool (100% agree). The software process engineers were partially positive about the functionality provided by the tool (50% neither agree nor disagree and 50% agree).

The software quality results support the suitability of the organizational context definition tool, indicating that this application can be used in industrial settings; the users *agree* with the quality factors of the tool (75%), however, the functionality of the system still needs some improvements.

We corroborated that the understandability of domain concepts are supported by other previously analyzed characteristics; particularly, usability (familiarity and simplicity) and operability (simplicity of the process). Moreover, the correctness of the service is supported by usability (consistency). On the other hand, the service usability is supported by the previously analyzed characteristics. Regardless of the positive results, the functionality aspect



	Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree
Understability of domain concepts	0%	0%	0%	100%	0%
Correctness of the tool	0%	0%	0%	100%	0%
Usability of the tool	0%	0%	0%	100%	0%
Functionality of the tool	0%	0%	50%	50%	0%

Figure 8.16: Distribution of software quality factors results according to the Likert scale.

of the organizational context definition service should be improved.

We analyzed the logbook for complementary information about the service functionality and identified several functionality limitations that can be attributed to *lack of flexibility for conducting basic operations*. In this case, the identification of functionality improvement aspect was supported by other previously analyzed factors; particularly by operability (flexibility). Next, we present some quotes retrieved from the logbook.

"More flexibility in the use of basic operations (save, edit, delete) can facilitate the organizational context definition".

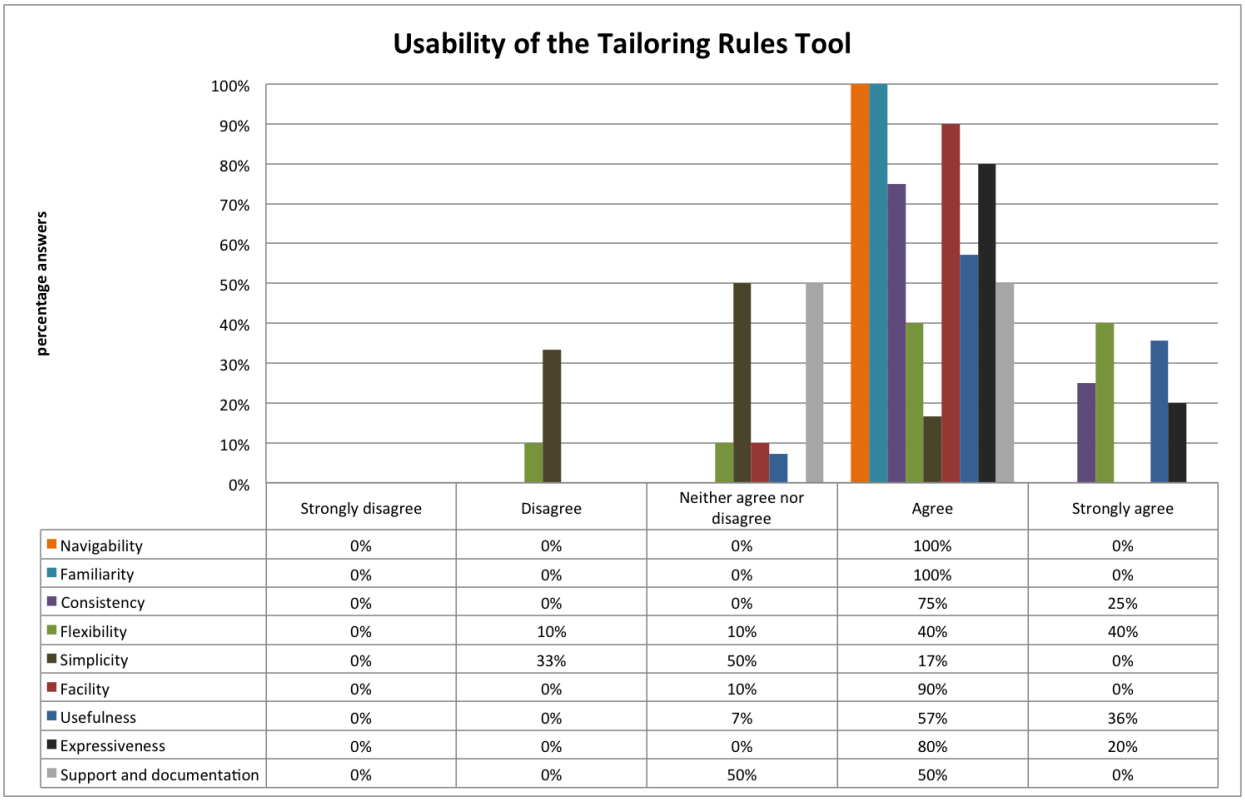
"Probably, the functionality of basic operations can improve in terms of the guidance provided to the users".

While these results indicate that the organizational context definition is quite usable and operable, we cannot positively conclude the existence of such an effect in other scenarios, because the observed characteristics are not statistically significant. However, the evidence suggests the organizational context definition service worked well in practice, because the users were able to define such a context in a usable manner using the domain concepts.

Evaluation of tailoring rules definition

In the evaluation of the tailoring rules definition service embedded in ATAGeTT, we considered several quality attributes of the software (related to usability) and also expressiveness that provide the user interfaces for defining the tailoring rules. This evaluation was performed by two software process engineers. Next we briefly describe the evaluated attributes.

Usability. The usability was evaluated in terms of navigability (2 sentences), familiarity



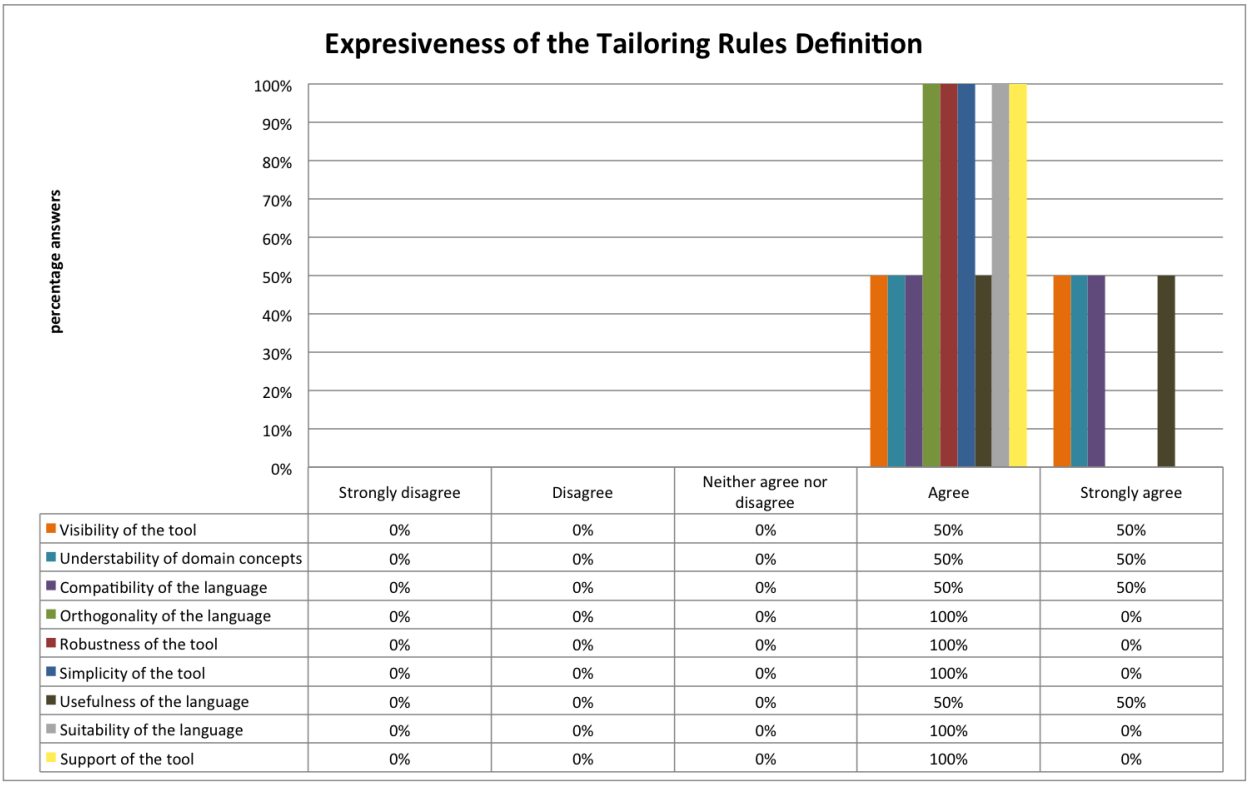
	Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree
Navigability	0%	0%	0%	100%	0%
Familiarity	0%	0%	0%	100%	0%
Consistency	0%	0%	0%	75%	25%
Flexibility	0%	10%	10%	40%	40%
Simplicity	0%	33%	50%	17%	0%
Facility	0%	0%	10%	90%	0%
Usefulness	0%	0%	7%	57%	36%
Expressiveness	0%	0%	0%	80%	20%
Support and documentation	0%	0%	50%	50%	0%

Figure 8.17: Distribution of usability results about the tailoring rules definition using a Likert scale.

(3 sentences), consistency (4 sentences), flexibility (5 sentences), simplicity (5 sentences), facility (5 sentences), usefulness (6 sentences), expressiveness (5 sentences) and support and documentation (3 sentences). Figure 8.17 presents the results of the usability evaluation. The software process engineers indicated that the tailoring rules definition tool was *usable* in terms of navigability (100% agree), familiarity (100% agree), consistency (75% agree and 25% strongly agree), flexibility (40% agree and 40% strongly agree), facility (90% agree), usefulness (57% agree and 36% strongly agree) and expressiveness (80% agree and 20% strongly agree). However, these evaluators were partially negative about the simplicity of the tool (33% disagree and 50% neither agree nor disagree), and partially positive about the tool support and documentation (50% neither agree nor disagree and 50% agree).

The survey results show a positive opinion about the tool usability and it offers a good user experience during the organizational context definition. Moreover, we can indicate that the tailoring rules definition tool is usable in practice because 78% of usability factors were evaluated between *Agree* and *Strongly agree*. However, the simplicity, support and documentation of the tool requires some improvements.

Following the same strategy as in previous cases, we analyzed the logbook to obtain complementary information about simplicity, support and documentation. The comments about simplicity can be attributed to either: (1) *Ease of use of the tool* and (2) *Flexibility of tailoring rules definition*. The comments about support and documentation of the tool that can be attributed to either: (1) *Availability of the helper information* and (2) *Examples*



	Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree
Visibility of the tool	0%	0%	0%	50%	50%
Understability of domain concepts	0%	0%	0%	50%	50%
Compatibility of the language	0%	0%	0%	50%	50%
Orthogonality of the language	0%	0%	0%	100%	0%
Robustness of the tool	0%	0%	0%	100%	0%
Simplicity of the tool	0%	0%	0%	100%	0%
Usefulness of the language	0%	0%	0%	50%	50%
Suitability of the language	0%	0%	0%	100%	0%
Support of the tool	0%	0%	0%	100%	0%

Figure 8.18: The percentage distribution (in terms of Likert items) of the expressiveness characteristics.

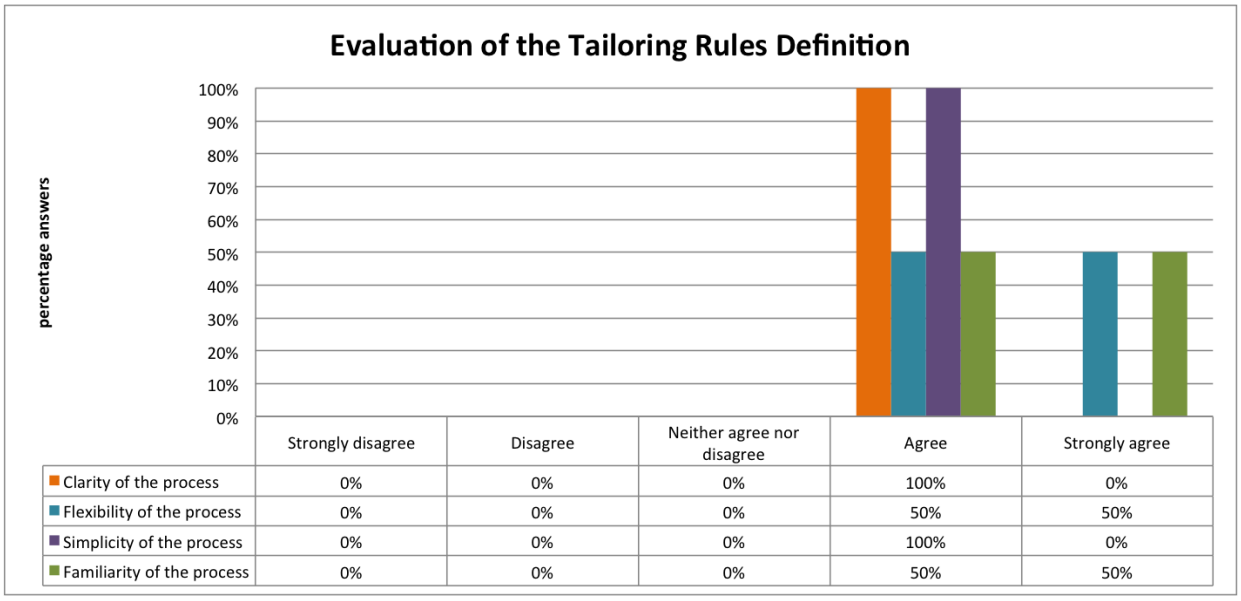
and documentation are required in other actions. Next, we present some quotes from the comments about these quality aspects made by process engineers:

"The process for selecting input XML files can be implemented in a more simple way".

"It is not easy to find help documentation about tailoring rules definitions".

"It is required more information about the helpers; for instance when using complex conditions".

Expressiveness. The expressiveness of the decision language was evaluated in terms of visibility, understandability, compatibility, orthogonality, robustness, simplicity, usefulness, suitability and support (9 sentences). Figure 8.18 presents the results of the expressiveness. The software process engineers *Agree* and *Strongly agree* that the tailoring rules definition service is expressive enough as to allow people to define the rules they required to tailor their process. This *Expressiveness* was reported through the visibility of the tool (50% agree and 50% strongly agree), understandability of domain concepts (50% agree and 50% strongly agree), compatibility of the language (50% agree and 50% strongly agree), orthogonality of the language (100% agree), robustness of the tool (100% agree), simplicity of the tool (100% agree), usefulness of the language (50% agree and 50% strongly agree), suitability of the language (100% agree) and support of the tool (100% agree). In this case, all expressiveness factors were evaluated between *Agree* and *Strongly agree*.



	Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree
Clarity of the process	0%	0%	0%	100%	0%
Flexibility of the process	0%	0%	0%	50%	50%
Simplicity of the process	0%	0%	0%	100%	0%
Familiarity of the process	0%	0%	0%	50%	50%

Figure 8.19: The percentage distribution (in terms of Likert items) of the software quality factors.

The survey results suggest that the decision language is expressive enough as for defining tailoring rules in terms of orthogonality, robustness, simplicity and suitability, because this language can express the intentions of the software process engineers, and allows them to represent simple and complex conditions. In addition, the graphical environment allows process engineers to use the decision language without interacting with the concrete syntax of the language that can be complex to understand. Therefore, we can suggest that the decision language can be used for end-users without MDE knowledge, using a tool support that hides the complexity of the language, since there is a binding between the decision language and the tool.

After analyzing the logbook to obtain complementary information about expressiveness, we do not identify improvement opportunities reported by the process engineers. We can therefore speculate that the decision language is easier to use than writing tailoring rules directly in a transformation language, such as ATL. Moreover, the decision language keeps the essential structure of the tailoring rules (conditions and conclusion) that can be used for generating a transformation without sacrificing expressiveness.

Software quality factors. Other quality factors of the tailoring rules definition service were evaluated in terms of clarity, flexibility, simplicity and familiarity (4 sentences). Figure 8.19 presents the results of this software quality evaluation. The software process engineers *Agree* and *Strongly agree* that this service has good *Quality* in terms of clarity (100% agree), flexibility (50% agree and 50% strongly agree), simplicity (100% agree) and familiarity (50% agree and 50% strongly agree) of the process. In this case, all expressiveness factors were evaluated between *Agree* and *Strongly agree*.

The obtained results suggest that the tailoring rules definition has good quality and it can be used in practice. In this sense, we can corroborate that the flexibility of the process is supported by other analyzed factors like usability (flexibility). Moreover, the familiarity of the

process is supported by both, usability (familiarity) and expressiveness (understandability). However, we can recognize that the tailoring rules definition requires improvements in terms of clarity and simplicity. Comments recorded in the logbook identify improvement opportunities that can be attributed to either: (1) *Simplicity for creating tailoring rules*, (2) Clarify for editing tailoring rules. In this case, the clarity and simplicity improvements were supported by other factors previously analyzed: usability (simplicity, support and documentation) and expressiveness (simplicity of the tool). Next, we present some quotes from the logbook:

"Although the tailoring rules definition works well in practice, it requires more simplicity for creating tailoring rules".

"More clarify for editing tailoring rules can help us to maintain them".

Finally, the evidence suggest that the tailoring rules definition is usable and useful enough, because the users can define tailoring rules in a quite simple manner. Moreover, the decision language is expressive enough because the users can define tailoring rules using the language constructors. Although the results are promising in terms of expressiveness and usability, we are aware that the reality of each software company may be also different, and therefore these results cannot be generalized to other software companies. We plan (as future work) to verify the expressiveness of the DL by replicating this study in other scenarios, but always considering small and medium-sized software companies.

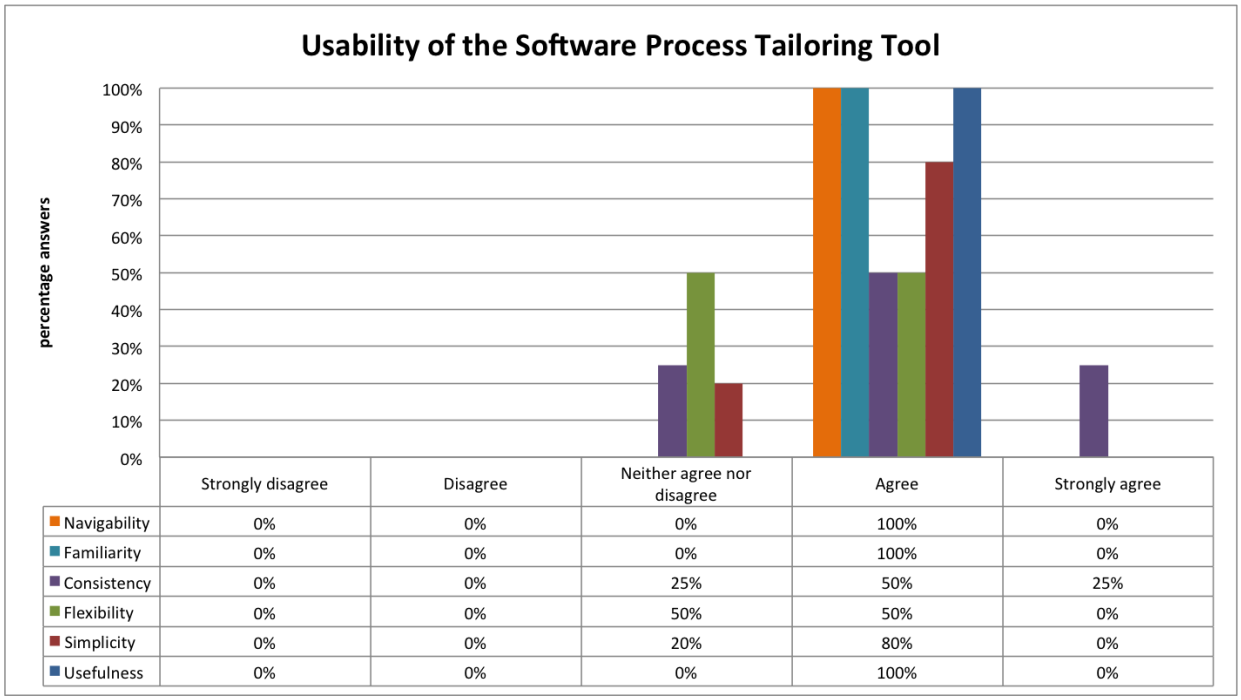
Evaluation of project context definition

In the evaluation of the project context definition service embedded in ATAGeTT we considered its usability, operability and the quality factors included in the project context survey. This evaluation were performed by one project manager since this is part of his business activity.

Usability. The usability of the service was evaluated in terms of navigability (2 sentences), familiarity (3 sentences), consistency (4 sentences), flexibility (5 sentences), simplicity (5 sentences), usefulness (5 sentences). Figure 8.20 presents the results of the usability evaluation of the project context definition tool. The project manager *Agrees* and *Strongly agrees* that the project context tool has factors that suggest good *Usability* of the service in terms of navigability (100% agree), familiarity (100% agree), consistency (50% agree and 25% strongly agree), simplicity (80% agree) and usefulness (100% agree). However, this person is partially positive about the flexibility (50% neither agree nor disagree and 50% agree) shown by the service.

The usability results show a positive tendency indicating that the user can learn and use the project context tool. This service offers a good user experience during the project context definition. However, the tool still has space for improving its flexibility in terms of *facility of the project context configuration.* The following are some quotes from the logbook:

"The usability of the project context definition tool is good enough, but the project context configuration should be more flexible to configure general contexts without considering a particular project".



	Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree
Navigability	0%	0%	0%	100%	0%
Familiarity	0%	0%	0%	100%	0%
Consistency	0%	0%	25%	50%	25%
Flexibility	0%	0%	50%	50%	0%
Simplicity	0%	0%	20%	80%	0%
Usefulness	0%	0%	0%	100%	0%

Figure 8.20: Software process of Ki teknology (EPF version)

Operability. The operability was evaluated in terms of flexibility, usefulness, correctness and simplicity (4 sentences). Figure 8.21 presents the results of the operability of the automatic tailoring. The project manager considered the automatic tailoring of software process operable in terms of flexibility (100% strongly agree), usefulness (100% agree), correctness (100% agree) and simplicity (100% strongly agree). He also thinks that the system reduces the time and effort needed for unplanned interventions.

This evidence suggests that this tailoring strategy provides not only reliable functionality to end-users, but also it works well with the common and complementary activities supported by ATAGeTT. This automatic tailoring has been designed and implemented to operate with artifacts that were generated by the organizational context and the tailoring rules definition tools.

Software quality factors. The software quality factors were evaluated in terms of understandability, correctness, usability and functionality (4 sentences). Figure 8.22 presents the results of the software quality evaluation. The project manager was highly positive about the *Quality* of the project context definition tool, in terms of the understandability of domain concepts (100% strongly agree), and the correctness (100% strongly agree), usability (100% agree) and functionality of the tool (100% agree).

We can corroborate that the understandability of domain concepts is supported by other factors analyzed: usability (familiarity) and operability (simplicity). The correctness of the tool is supported by usability (consistency) and operability (correctness). The usability is supported by usability factors, and the functionality is supported by operability factors. On the other hand, we recognize that the flexibility of the project context definition tool should improve its usability.

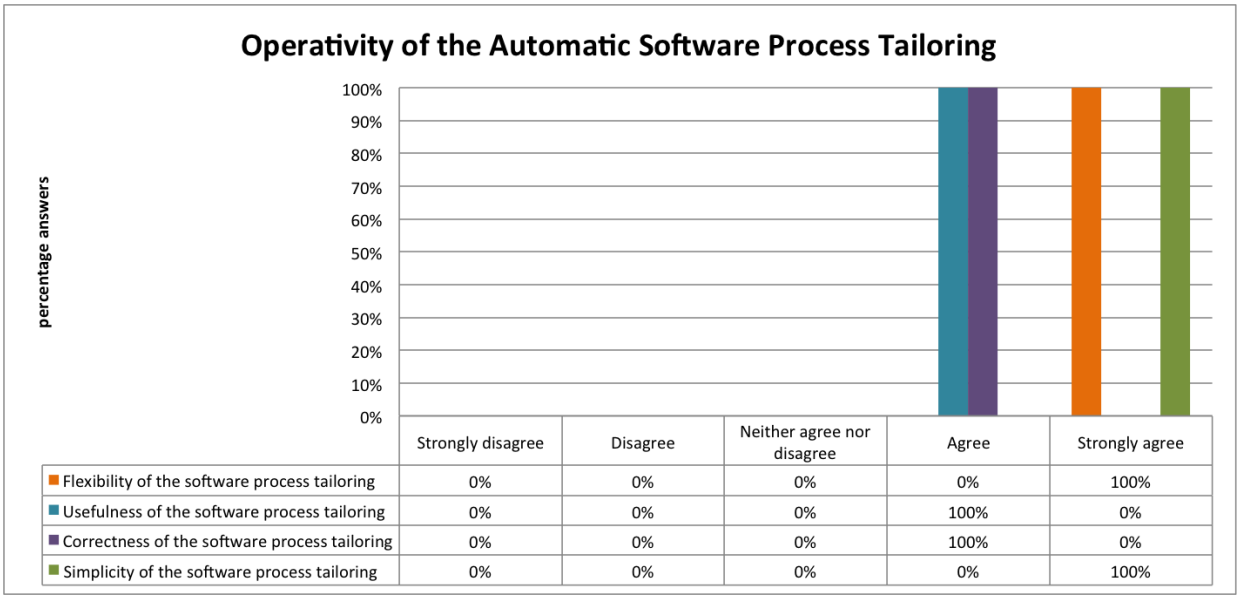


Figure 8.21: Software process of Ki teknology (EPF version)

The evaluation results show that this tool automatically generates transformations using the decision language for defining tailoring rules, and applies the generated transformation for obtaining an adapted software process. Furthermore, using the tool the Ki agile process can be adapted to different project contexts in a usable and transparent manner for the project managers.

8.5.3.3 About usefulness of ATAGeTT

In order to understand how the use of ATAGeTT impacted the software process tailoring of Ki teknology, we considered one closed-ended question that can be answered by a simple "yes" or "no", and also one open-ended question. The first one evaluates the usefulness of the tool, and the second one intends to identify improvement aspects for this application. Next, we present the answers to each question.

Concerning the organizational context definition tool

In the evaluation of the project context definition, the answers to the questions were the following:

Do you consider that the tool is useful for your software company?

The two software process engineers answered "yes", showing that the organizational context tool was useful for defining such the organizational context.

What would you suggest to improve the user experience when using the tool?

In this case the process engineers suggested the following improvements: *"Increasing the*

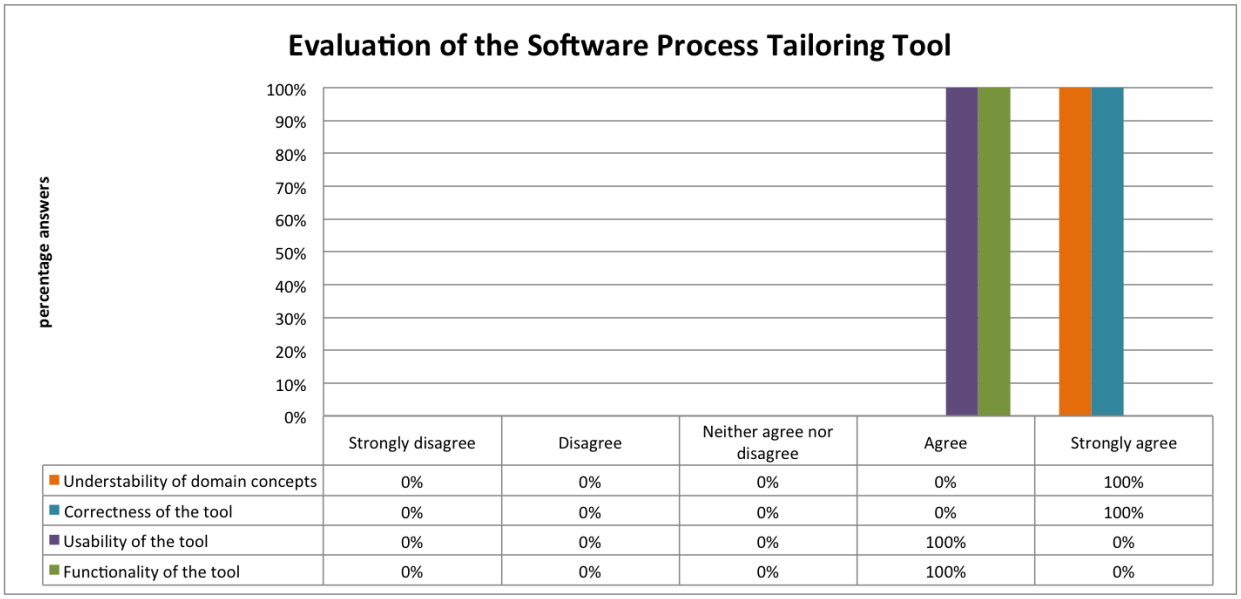


Figure 8.22: Software process of Ki teknology (EPF version)

margins of the user interface", *"Improving the navigability of the user interface"*, *"Improving the flexibility of the tool for defining optional context attributes"*.

These answers suggest a correlation with the evaluation of the usability and operability of the organizational context definition, where usability (navigability), operability (flexibility) and software quality (functionality) required improvements.

Concerning the tailoring rules definition tool

Next we present the answers to the questions that evaluated the suitability of the tailoring rules definition tool:

Do you consider the tool useful for supporting the tailoring rules definition?

In this case the two process engineers answered "yes", emphasizing that the tool eases the definition of tailoring rules and the generation of the tailoring transformation.

What would you suggest for improving the user experience while using the tool for tailoring rules definition?

Here the process engineers indicated: *"There is a need to improve the support and documentation of the tool. It also needs to include help buttons or pop-up information indicating the meaning of the user interface elements"* and *"It requires more flexibility for editing the variable process elements that have tailoring rules"*. These comments show a correlation with the evaluation of the usability and expressiveness of the tailoring rules definition, where usability (simplicity, support and documentation) has space for improvements.

Concerning the project context definition tool

Next we indicate the questions used in the evaluation of the project context definition tool, and also the corresponding answers:

Do you consider the tool useful for supporting the project context definition?

In this case the project manager (the only evaluator) answered "yes", indicating the application was useful for generating a project context and executing the software process tailoring.

Are you willing to adopt the automatic software process tailoring based on project context and tailoring rules? Why?

The project manager answered *"Yes. Today we do it manually. If we can automatically obtain a software process adjusted to the characteristics of the project to address, we should improve the efficiency of our developments by reducing the operational costs"*. These answers suggest a potential suitability of ATAGeTT for being adopted by the software industry. Moreover, the project manager recognizes potential benefits of adapting the Ki agile process for reducing their operational costs.

8.6 Discussion

In order to understand the evaluation results of this chapter, next we summarize the data analysis of this case study.

- **Timing**. The software process engineers required between 3.18 and 4.32 minutes for defining the organizational context that considers six context attributes; and they required between 13.40 and 14.10 minutes for defining the tailoring rules that considers seventeen rules. These numbers make us expect a good user experience in industrial settings. On the other hand, the project manager required between 1.35 minutes for defining the project context used to adapt the organizational software process. Although one sample is clearly not enough to make conclusions, the number is highly positive.
- **Organizational context definition**. According to the preliminary results, the organizational context definition tool was usable in this industrial setting; a 83% of usability factors were evaluated between *Agree* and *Strongly agree*. The organizational context definition was operable; a 75% of operability factors were evaluated between *Agree* and *Strongly agree*. The quality of the organizational context definition tool was good enough as to be used in industrial settings; the users evaluated positively 75% of the quality factors.
- **Tailoring rules definition**. The tailoring rules definition tool offers a good user experience because 78% of usability factors were evaluated between *Agree* and *Strongly agree*. The decision language resulted expressive enough as to define tailoring rules of the company. The language was able to express the intentions of the software process

engineers and 100% of expressiveness factors were evaluated between *Agree* and *Strongly agree*. The rules definition tool was perceived as with good quality, and therefore it could be used in practice.

- **Project context definition**. The project context definition tool was also perceived as usable enough. The user thinks that he can (and also other project managers) learn and use this tool in practice. The automatic software process tailoring was perceived as operable because it works well with the common (mandatory) and complementary activities. Moreover, the tool had also enough quality as to use it in practice. The software quality factors for applying in the practice and also the integration with other artifacts of the ATAGeTT did not show compatibility problems.

On the other hand, in this study we initially formulated two research questions for evaluating ATAGeTT and the decision language:

- **How well does ATAGeTT work in practice?** The evidence suggests that the three tools embedded in ATAGeTT offer a good user experience and are useful in the evaluated industrial settings; therefore, they could be used in practice. The system provides an usable user interface that allows its users -the software process engineers and the project managers- to deal only with software process and context-related concepts, hiding the complexity of manipulating the models and transformations involved in the tailoring activity.
- **How expressive is the decision language for specifying transformation rules to tailor software processes?** The evidence suggests that the decision language is expressive enough to specify tailoring rules, and the language constructs can express the intentions of the process engineers. Moreover, using the decision language is easier than writing the tailoring rules directly in a transformation language, because in the first case the use of the language is hidden behind a user-friendly graphical user interface.

These preliminary results are promising. Nevertheless, we are aware that the reality of software companies may be diverse, and therefore the suitability of ATAGeTT is not guaranteed in all scenarios. For instance, it may be the case that some conditions for resolving variability do not only depend on context values, but also on the way that other variation points are resolved during the tailoring. The current version of the tool does not support these kinds of conditions yet, but we have not found the need of doing it in practice.

Finally, the answers to the two research questions allow us to expect a positive adoption of ATAGeTT in at least a portion of the software industry, since the end-users recognize the potential benefits of automatically adapting their software processes, reducing thus operational costs and improving the competitiveness of the software companies. The use of this solution will probably also enhance the quality of the obtained software products, but this evaluation is out of this thesis scope.

The obtained results are not strong enough to make conclusions given the small number of users participating in the case studies; therefore, we cannot generalize the results to other software companies. However, the results are highly consistent, which make us expect positive experiences when the ATAGeTT are applied for tailoring processes in other software companies.

8.7 Threats to Validity

The threats to the validity of this explanatory case study are analyzed according to the aspects proposed in [173]: construct validity, reliability, internal validity, and external validity.

8.7.1 Construct validity

This aspect of validity is concerned with the extent to which the concepts being studied, really represent what the researcher claims to study. We have worked with MDE-based software process tailoring for the last six years, and this approach has proved to be technically feasible. However, MDE-based tailoring is difficult to deploy in industrial settings because it requires knowledge for writing transformations and manipulating models. For that reason we developed a generic and usable tool for automatically generating transformations and applying MDE-based software process tailoring. The preliminary results show ATAGeTT improved the usability of the current solutions for transformation generation, through a graphical user interface that includes only domain concepts.

8.7.2 Reliability

This aspect is concerned with the fact that the case study can be repeated by other researchers, obtaining the same or a similar result. In this sense, we provide the set of ATAGeTT components that allows other people to tailor software processes step-by-step, through the definition of organizational contexts, tailoring rules and project contexts. In consequence, other researchers would have no difficulty in applying ATAGeTT for software process tailoring in the same or other software organizations. This tool is available online at: http://adapte.dcc.uchile.cl:8080/Context.

8.7.3 Internal validity

This aspect is concerned with how well our case study was conducted. A source of threats to the internal validity would be the reliance on the focus group conducted with the aim of reviewing the software processes, organizational contexts and tailoring rules of Ki teknology. Although such a reviewing process was guided by a case study protocol, it was necessary to have a certain level of trust in the information about the software process (including tasks, roles and work products), organizational context (including context attributes and their values) and tailoring rules of the company (conditions and conclusions). However, the effect of this aspect was minimized using different sources of evidence, like surveys, observations and automatic records generated by ATAGeTT (log files).

On the other hand, the validation of ATAGeTT focused on evaluating usability and expressiveness. In this sense, there may be a threat related to the evaluation of user interfaces.

However, this aspect was minimized considering other quality factors that were part of the survey forms, such as: clarity, flexibility, simplicity, familiarity and functionality.

8.7.4 External validity

The external validity is concerned with the generalization of the findings. The validation of ATAGeTT was conducted in small and medium-sized Chilean software companies. Although, the validation results are promising, we are aware that the processes and tailoring strategies, e.g., of large software companies, may be more complex. Therefore, the suitability of ATAGeTT is not guaranteed and the results obtained in the case studies are not generalizable to a broader population; even if we think only about the Chilean software industry.

Chapter 9

Conclusions, Contributions, Limitations and Future Work

9.1 Summary of the Thesis Work

In order to systematize development, software companies define their organizational software processes. A process engineer is usually in charge of this activity. Tailoring the organizational software processes is an activity that allows project managers to adapt such general processes to the needs and characteristics of a particular project. MDE-based tailoring strategies have been applied with that purpose using models and transformations. However, building appropriate models and transformations requires expertise of the process engineers for specifying tailoring rules, making decisions, choosing the right kind of transformation, and also mastering the transformation language syntax. In this sense, there are potential challenges in terms of usability, adoption, and evolution of the MDE-based software process tailoring that are not easy to address, mainly by small and medium-sized software companies.

To address these challenges, Chapter 3 presents several tools for defining organizational context models and project context models through a graphical user interface without interacting with models and metamodels. These tools reduce some difficulties for generating the input models that MDE-based software process tailoring required. Moreover, we also created projectors that allow users to translate software processes into software process models, and vice versa. The translation back and forth is now transparent for the end-users, and therefore these people can easily understand and validate the tailored software process. However, writing tailoring transformations and their rules remained a high challenge for the process engineers.

In order to improve the support for MDE-based software process tailoring, we presented a DSL for specifying tailoring rules (Chapter 4) and a HOT for automatically generating tailoring transformations (Chapter 5). The DSL is a decision language that allows process engineers to formally specify tailoring rules as a model, and the HOT is a generic transformation that takes the previous model and automatically generates a tailoring transformation. Moreover, the DSL and HOT are supported by a graphical user interface that improves the

user experience by hiding the complexity of applying an automatic MDE-based strategy for tailoring a software process.

The tools and the graphical environments were integrated in a tool-set named ATAGeTT (Chapter 6). For tailoring a software process the users of this system only require to know the organizational context, project characteristics and how they affect software process tailoring. In order to evaluate the suitability of ATAGeTT we conducted an exploratory case study, and then an explanatory case study validating the results.

In Chapter 7 the exploratory case study compared advantages and disadvantages of using a template-based and an automatic MDE-based strategies for tailoring a software process. The comparison between these two strategies was done in terms of productivity and correctness (**RQ1: Considering productivity and correctness, what is the difference between the tailored software processes obtained using each strategy?**). The results indicated that the automatic MDE-based tailoring is always more productive and correct for generating a software process for each project context than the template-based approach.

On the other hand, in Chapter 8, the explanatory case study validated the usability of ATAGeTT and expressiveness of the decision language (**RQ2: How well does ATAGeTT work in practice? and RQ3: How expressive is the decision language for specifying transformation rules to tailor software processes?**). The evidence suggests that ATAGeTT offers good user experience and it is useful for the end-users for tailoring software processes. Moreover, the decision language is expressive enough for specifying tailoring rules, and the language constructs can express the intentions of software process engineers. Although the results are promising, more empirical evidence is required to obtain stronger conclusions.

9.2 Conclusions

Considering the problem described in Chapter 1, we have stated three hypotheses:

H1: The complexity of using the MDE-based tailoring approach can be hidden through the use of a high-level language that allows defining tailoring rules using just process concepts and project characteristics.

H2: The MDE-based tailoring approach allows automatic generation and execution of the tailoring transformations that software companies may require.

H3: The whole MDE-based tailoring approach can be implemented as a usable organization-independent tool.

Regarding (H1), we proposed a decision language for specifying tailoring rules that is supported by a graphical user interface that hides the complexity of directly manipulating the language. Moreover, the evidence of the explanatory case study suggests that the decision language is expressive and suitable enough as for defining tailoring rules in the software process domain.

Concerning (H2), we proposed a Higher-order Transformation (HOT) and ATL extractor for generating tailoring transformations that are supported by another graphical user interface. The evidence of the exploratory and the explanatory case studies suggests that the MDE-based tailoring approach is useful for generating a software process tailored according to each project context. In addition, the graphical user interface generates and executes in a usable manner, the tailoring transformations required by the software companies according to the previous specifications.

Concerning (H3), we presented ATAGeTT for encapsulating the complexity of the tailoring rules specification and the tailoring transformation. The evidence collected during the explanatory case study suggests that ATAGeTT is usable and it can be used in practice by other software companies. The tool is available online (http://adapte.dcc.uchile.cl:8080/Context), therefore any researcher can use it for applying the proposed MDE-based software process tailoring approach.

After using ATAGeTT in three software companies and obtaining positive results in practice, we can say that the use of the tool is feasible even for small software companies. The effort of using ATAGeTT involves the definition of the organizational context, the tailoring rules and the project context, and as a result the project manager obtains a tailored software process adjusted to the project features.

These results support the general research question defined in Section 1.3.1: *RQ: Can software process tailoring be successfully conducted in an expert-independent way?*. Considering the previously mentioned restrictions, we can say that the case studies results are aligned with the stated hypotheses, since ATAGeTT has shown to be usable and useful for applying the MDE-based software process tailoring through a graphical user interface. In this sense, the users no longer require expertise in MDE for generating models or transformations, because ATAGeTT hides such a formality behind its user interface.

9.3 Contributions

This thesis makes three contributions to the state-of-the-art: (1) a formal specification of tailoring rules using models, (2) a systematic generation of tailoring transformations using HOTs, (3) a software tool-set that supports the tailoring rules specification and the automatic tailoring transformations generation and execution.

Concerning the first contribution, we defined a DSL (called Decision Language) that supports the specification of tailoring rules. This language was specified on the notion of modelware, and we defined a variation decision model that conforms to a variation decision metamodel. The variation decision model is a formal specification of tailoring rules that was inspired in decision models and semantics of business vocabulary and business rules used for building decision rules. The proposed decision language reduces the complexity of defining tailoring rules by just using domain concepts: process and context elements. Finally, the variation decision metamodel is freely available and it can be downloaded from http://www.dcc.uchile.cl/∼lsilvest/DSL/VDMM.ecore (VDMM is compatible with Eclipse

Modeling Framework).

Concerning the second contribution, we defined a HOT (called VDM2ATL) that automatically generates tailoring transformations. The HOT takes the variation decision model and generates a tailoring transformation. However, this is achieved in two steps: (1) the HOT takes the variation decision model and generates a transformation model using a model-to-model transformation, and (2) the ATL extractor takes the transformation model and generates the tailoring transformation as ATL code, using a model-to-text transformation. The developed HOT is generic, therefore, it can be used in any company regardless the software process, project context and tailoring rules specifically defined by the software company. Such an application is freely available at: http://www.dcc.uchile.cl/∼lsilvest/HOT/VDM2ATL.atl (VDM2ATL is compatible with ATL Integrated Environment).

The third contribution is the integrated tool (ATAGeTT) that was designed for defining tailoring rules, generating tailoring transformations and executing MDE-based software process tailoring. This tool allows software process engineers to interactively define tailoring rules using the decision language and generate transformations using the HOT, taking advantage of the formality provided by MDE, but hiding its inherent complexity behind a user interface. ATAGeTT allows project managers to execute the MDE-based software process tailoring in a transparent way, and it has shown to be usable and useful in the companies that participated in the case studies. We expect these results can also be obtained by other software companies in future evaluation experiences of the tool.

From the practical experience with the industrial partners, we have not found a tailoring rule that could not be specified using the decision language, raising thus our confidence about its expressiveness for this application domain. Moreover, we have generated tailoring transformations for three software companies in a correct and automatic way, raising also our confidence on the reusability of the proposed HOT. Finally, we have validated that the proposed integrated tool is easy to learn and use for the potential users –software process engineers and project managers–, raising also our confidence on the usability and usefulness of ATAGeTT. We expect these contributions ease the tailoring of software processes, particularly in small and medium-sized organizations that are usually limited in performing this activity.

9.4 Scope of the Proposed Solution

The proposed solution requires that the software companies have their organizational software processes specified in Eclipse Process Framework (EPF). In this sense, ATAGeTT does not work with others specifications.

The decision language was designed for defining tailoring rules in the software process domain, and particularly in small and medium-size Chilean software companies. Therefore, the expressiveness of the language cannot be guaranteed in other application domains or other type of companies.

Concerning the HOT, it is an important support for the development of model transformations in the software process domain; however, we recognize that this component does not work with other input models, and generates only transformations as ATL code.

Although ATAGeTT offers good user experience and can be used in practice in small and medium-sized software companies, it may be not suitable for tailoring software processes in large software companies, since the processes and the tailoring are usually much more complex than in the former.

9.5 Future Work

ATAGeTT was validated in three Chilean software companies, however it is not enough to get strong conclusions about the generality of its usability and usefulness. Therefore, we intend to validate the tool in other software companies both in Chile and Bolivia. These experiences will help us identify future improvements, particularly for improving its capability to be adopted by small and medium-sized software organizations.

As part of the future work we also want to improve the use of ATAGeTT by connecting other software processes modeling tools that support a separation of concerns between processes and methods using SPEM (such as Enterprise Architect and Rational Method Composer). In this sense, we have to review the software process modeling tools that are used in the software industry (including those used by large software companies).

Concerning the decision language, we also want to extend it for including negative conditions using the logical operator NOT. The logical operator NOT allows the users to express conditions that are best expressed in a negative way. Although this operator was not required in the presented case studies, some conditions would have been simpler if the language had counted on the NOT operator.

We also plan to use weaving models for improving the consistency between the models used as input for the VDM (i.e., the organizational software process and the organizational context model) and software process and project context used as input for the generated tailoring transformation. In the current implementation of ATAGeTT this consistency is assumed but not ensured. Weaving models can be used for managing traceability between common elements that are defined in the organizational software processes, organizational contexts and tailoring rules.

Bibliography

[1] Silvia Abrahao and Emilio Insfran. Early usability evaluation in model driven architecture environments. In *Quality Software, 2006. QSIC 2006. Sixth International Conference on*, pages 287–294. IEEE, 2006.

[2] Silvia Teresita Acuña and Xavier Ferré. Software process modelling. In *World Multiconfer- ence on Systemics, Cybernetics and Informatics, ISAS-SCIs*, volume 1, pages 237–242, 2001.

[3] Ove Armbrust, Masafumi Katahira, Yuko Miyamoto, Jürgen Münch, Haruka Nakao, and Alexis Ocampo. Scoping software process lines. *Software Process: Improvement and Practice*, 14(3):181–197, 2009.

[4] Daniel Balasubramanian, Anantha Narayanan, Chris vanBuskirk, and Gabor Karsai. The graph rewriting and transformation language: GReAT. *Electronic Communications of the EASST*, 1, 2007.

[5] Krishnakumar Balasubramanian, Aniruddha Gokhale, Gabor Karsai, Janos Sztipanovits, and Sandeep Neema. Developing applications using model-driven design environments. *Computer*, 39(2):33–40, 2006.

[6] Mikaël Barbero, Frédéric Jouault, and Jean Bézivin. Model driven management of complex systems: Implementing the macroscope's vision. In *Engineering of Computer Based Systems, 2008. ECBS 2008. 15th Annual IEEE International Conference and Workshop on the*, pages 277–286. IEEE, 2008.

[7] Ankica Barišić, Vasco Amaral, Miguel Goulao, and Bruno Barroca. Quality in use of domain-specific languages: a case study. In *Proceedings of the 3rd ACM SIGPLAN workshop on Evaluation and usability of programming languages and tools*, pages 65–72. ACM, 2011.

[8] Victor Basili and Dieter Rombach. Tailoring the Software Process to Project Goals and Environments. In *Proceedings of the 9th International Conference on Software Engineering*, ICSE '87, pages 345–357, Los Alamitos, CA, USA, 1987. IEEE Computer Society Press.

[9] Kent Beck, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, et al.

Manifesto for agile software development. 2001.

[10] Aldo Bertero, Luis Silvestre, and María Cecilia Bastarrica. T2m and m2t transformations between software processes and software process models. In *SCCC 2012, Proceedings of the XXXI International Conference of the Chilean Computer Science Society, Valparaiso, Chile, November 2012*, pages 36–40. IEEE, 2012.

[11] Jean Bézivin. Model driven engineering: an emerging technical space. In Heidelberg Berlin, editor, *Proceedings of the 2005 International Conference on Generative and Transformational Techniques in Software Engineering*, GTTSE'05, pages 36–64, Braga, Portugal, 2006. Springer-Verlag.

[12] Jean Bézivin, Fabian Büttner, Martin Gogolla, Frédéric Jouault, Ivan Kurtev, and Arne Lindow. Model transformations? transformation models! In Nierstrasz et al. [105], pages 440–453.

[13] Jean Bézivin, Guillaume Hillairet, Frédéric Jouault, Ivan Kurtev, and William Piers. Bridging the ms/dsl tools and the eclipse modeling framework. In *Proceedings of the International Workshop on Software Factories at OOPSLA*, volume 5, 2005.

[14] Jean Bézivin, Frédéric Jouault, and Patrick Valduriez. On the need for megamodels. In *Proceedings of the OOPSLA/GPCE: Best Practices for Model-Driven Software Development workshop, 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications*, 2004.

[15] Pedro Borges, Paula Monteiro, and RicardoJ. Machado. Mapping RUP Roles to Small Software Development Teams. In Stefan Biffl, Dietmar Winkler, and Johannes Bergsmann, editors, *Software Quality. Process Automation in Software Development*, volume 94 of *Lecture Notes in Business Information Processing*, pages 59–70. Springer Berlin Heidelberg, 2012.

[16] Goetz Botterweck, Liam O'Brien, and Steffen Thiel. Model-driven derivation of product architectures. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, pages 469–472. ACM, 2007.

[17] Goetz Botterweck, Andreas Polzer, and Stefan Kowalewski. Using higher-order transformations to derive variability mechanism for embedded systems. In *Models in Software Engineering*, pages 68–82. Springer, 2009.

[18] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. Model-driven software engineering in practice. *Synthesis Lectures on Software Engineering*, 1(1):1–182, 2012.

[19] Marco Brambilla, Piero Fraternali, and Massimo Tisi. WebRatio Framework - WebML. *http* : *//hom*e.dei*b.poli*mi.it*/mbrambil/legacytom*da*/*.

[20] Marco Brambilla, Piero Fraternali, and Massimo Tisi. A metamodel transformation framework for the migration of WebML models to MDA. In *MDWE, CEUR Workshop Proceedings*, volume 389, pages 91–105. Citeseer, 2008.

[21] Marco Brambilla, Piero Fraternali, and Massimo Tisi. A transformation framework to bridge domain specific languages to MDA. In *Models in Software Engineering*, pages 167–180. Springer, 2008.

[22] Erwan Breton and Jean Bézivin. Model driven process engineering. In *COMPSAC*, pages 225–230. IEEE, 2001.

[23] Cédric Brun and Alfonso Pierantonio. Model differences in the eclipse modeling framework. *UPGRADE, The European Journal for the Informatics Professional*, 9(2):29–34, 2008.

[24] Javier Luis Cánovas, Jesús Sánchez, and Jesús Garcıa. Gra2mol: A domain specific transformation language for bridging grammarware to modelware in software modernization. In *Workshop on Model-Driven Software Evolution*, MoDSE'08, Athens, Greece, 2008.

[25] Antonio Cicchetti, Davide Di Ruscio, and Alfonso Pierantonio. A Metamodel Independent Approach to Difference Representation. *Journal of Object Technology*, 6(9):165–185, 2007.

[26] James Clark et al. Xsl transformations (xslt). *World Wide Web Consortium (W3C). URL http://www. w3. org/TR/xslt*, page 103, 1999.

[27] Paul Clarke and Rory V O'Connor. The situational factors that affect the software development process: Towards a comprehensive reference framework. *Information and Software Technology*, 54(5):433–447, 2012.

[28] Paul Clements and Linda Northrop. Software product lines: practices and patterns. 2002.

[29] Alistair Cockburn. *Crystal Clear a Human-powered Methodology for Small Teams*. Addison-Wesley Professional, first edition, 2004.

[30] Kieran Conboy and Brian Fitzgerald. Method and Developer Characteristics for Effective Agile Method Tailoring: A Study of XP Expert Opinion. *ACM Trans. Softw. Eng. Methodol.*, 20(1):2:1–2:30, July 2010.

[31] Krzysztof Czarnecki and Ulrich W Eisenecker. Generative programming. *Edited by G. Goos, J. Hartmanis, and J. van Leeuwen*, page 15, 2000.

[32] Krzysztof Czarnecki and Simon Helsen. Classification of Model Transformation Approaches. In *Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture*, volume 45, pages 1–17, Seattle, USA, 2003.

[33] Krzysztof Czarnecki and Simon Helsen. Feature-based survey of model transformation approaches. *IBM Systems Journal*, 45(3):621–646, 2006.

[34] Marcos Didonet Del Fabro and Jean Bézivin. Generic model management: from theory

to practice. In *First Intl. Workshop on Towers of Models*, 2007.

[35] 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*, pages 863–881. Springer, 2006.

[36] Marcos Didonet Del Fabro, Jean Bézivin, and Patrick Valduriez. Weaving Models with the Eclipse AMW plugin. In *Eclipse Modeling Symposium*, volume 2006. Citeseer, 2006.

[37] Hüseyin Ergin. *Design Patterns for Model Transformations*. PhD thesis, The University of Alabama, 2014.
http : *//hergin.stu*dents.cs.ua.edu/*research/proposal.*p*df*.

[38] Jean Rémy Falleri, Marianne Huchard, and Clémentine Nebut. Towards a traceability framework for model transformations in kermeta. In *Proceedings ECMDA Traceability Workshop (ECMDA-TW'06)*, pages 31–40, Bilbao, Spain, 2006.

[39] Patrick Farail, Pierre Gaufillet, Agusti Canals, Christophe Le Camus, David Sciamma, Pierre Michel, Xavier Crégut, and Marc Pantel. The TOPCASED project: a toolkit in open source for critical aeronautic systems design. *Embedded Real Time Software (ERTS)*, 781:54–59, 2006.

[40] Peter H Feiler and Watts S Humphrey. Software process development and enactment: Concepts and definitions. In *Software Process, 1993. Continuous Software Process Improvement, Second International Conference on the*, pages 28–40. IEEE, 1993.

[41] Matthias Felleisen. On the Expressive Power of Programming Languages. In *Science of Computer Programming*, pages 134–151. Springer-Verlag, 1990.

[42] Anthony Finkelstein, Jeff Kramer, and Bashar Nuseibeh. *Software process modelling and technology*. John Wiley & Sons, Inc., 1994.

[43] International Organization for Standardization and the International Electrotechnical Commission. *ISO/IEC 9126-1. Software engineering – Product quality*. ISO/IEC, 2001.

[44] Martin Fowler. *Domain-Specific Languages*. Addison Wesley, 2010.

[45] Robert France and Bernhard Rumpe. Model-driven development of complex software: A research roadmap. In *2007 Future of Software Engineering*, pages 37–54. IEEE Computer Society, 2007.

[46] Alfonso Fuggetta and Elisabetta Di Nitto. Software process. In *Proceedings of the on Future of Software Engineering*, pages 1–12. ACM, 2014.

[47] Kelly Garcés, Frédéric Jouault, Pierre Cointe, and Jean Bézivin. Adaptation of Models to Evolving Metamodels. Research Report RR-6723, INRIA, 2008.

[48] Kelly Garcés, Frédéric Jouault, Pierre Cointe, and Jean Bézivin. A domain specific lan-

guage for expressing model matching. In *Proceedings of the 5ère Journée sur l'Ingénierie Dirigée par les Modèles (IDM09)*, pages 33–48, 2009.

[49] Jokin García, Maider Azanza, Arantza Irastorza, and Oscar Díaz. Testing mofscript transformations with handymof. In *ICMT*, pages 42–56. Springer, 2014.

[50] Roxana Giandini, Gabriela Pérez, and Claudia Pons. Un lenguaje de Transformación específico para Modelos de Proceso del Negocio. In *XXXVI Conferencia Latinoamericana de Informática (CLEI 2010)*, volume 18, 2010.
http : *//www.lifia.info.unlp.edu.ar/eclipse/BPMTL/*.

[51] Bas Graaf and Arie Van Deursen. Using mde for generic comparison of views. Technical report, Delft University of Technology, Software Engineering Research Group, 2007.
http : *//www.eclipse.org/atl/atlTransformations/#KM32CONFATL*.

[52] Jeff Gray, Yuehua Lin, and Jing Zhang. Automating change evolution in model-driven engineering. *Computer*, 39(2):51–58, 2006.

[53] Jack Greenfield and Keith Short. Software factories: assembling applications with patterns, models, frameworks and tools. In *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 16–27. ACM, 2003.

[54] Object Management Group. Semantics of Business Vocabulary and Business Rules (SBVR) Version 1.0. Technical Report 2008-04-01, Object Management Group, 2008. http://www.omg.org/spec/SBVR/1.0/.

[55] Object Management Group. Software & Systems Process Engineering Metamodel Specification (SPEM) Version 2.0. Technical Report 2008-04-01, Object Management Group, 2008. http://www.omg.org/spec/SPEM/2.0/PDF/.

[56] Object Management Group. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. Technical report, Object Management Group, January 2011. http://www.omg.org/spec/QVT/1.1/PDF/.

[57] Regina Hebig, Holger Giese, Florian Stallmann, and Andreas Seibel. On the complex nature of mde evolution. In *International Conference on Model Driven Engineering Languages and Systems*, pages 436–453. Springer, 2013.

[58] Wolfgang Heider, Rick Rabiser, Deepak Dhungana, and Paul Grünbacher. Tracking evolution in model-based product lines. In *Proceedings 1st Int'l Workshop on Model-driven Approaches in Software Product Line Engineering (MAPLE 2009), collocated with the 13th Int'l Software Product Line Conference (SPLC 2009), San Francisco, CA, USA, August 24*, 2009.

[59] Guillaume Hillairet, Frédéric Bertrand, and Jean-Yves Lafaye. MDE for publishing data on the Semantic Web. *Transformation and Weaving Ontologies in MDE*, 395:32–46, 2008.

[60] Watts S Humphrey. Managing the software process (hardcover). *Addison-Wesley Professional. Humphrey, WS, & Curtis, B.(1991). Comments ona critical look'[software capability evaluations]. Software, IEEE*, 8(4):42–46, 1989.

[61] Watts S Humphrey. The software engineering process: definition and scope. *ACM SIGSOFT Software Engineering Notes*, 14(4):82–83, 1989.

[62] Julio Ariel Hurtado. *A meta-process for defining adaptable software processes*. PhD thesis, University of Chile, Facultad de Ciencias Físicas y Matemáticas, Santiago, Chile, 5 2012. http://www.repositorio.uchile.cl/handle/2250/111945.

[63] Julio Ariel Hurtado, María Cecilia Bastarrica, Sergio F Ochoa, and Jocelyn Simmonds. Mde software process lines in small companies. *Journal of Systems and Software*, 2012.

[64] Julio Ariel Hurtado, María Cecilia Bastarrica, Alcides Quispe, and Sergio Ochoa. MDE-Based Process Tailoring Strategy. *Journal of Software: Evolution and Process*, 2013.

[65] Julio Ariel Hurtado, María Cecilia Bastarrica, Alcides Quispe, and Sergio F. Ochoa. An MDE approach to software process tailoring. In David Raffo, Dietmar Pfahl, and Li Zhang, editors, *ICSSP*, pages 43–52, Honolulu, HI, USA, 2011. ACM.

[66] Julio Ariel Hurtado, María Cecilia Bastarrica, Alcides Quispe, and Sergio F. Ochoa. MDE-based process tailoring strategy. *Journal of Software: Evolution and Process*, 26(4):386–403, 2014.

[67] John Hutchinson, Jon Whittle, and Mark Rouncefield. Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure. *Science of Computer Programming*, 89:144–161, 2014.

[68] Tuomas Ihme, Minna Pikkarainen, Susanna Teppola, Jukka Kääriäinen, and Olivier Biot. Challenges and industry practices for managing software variability in small and medium sized enterprises. *Empirical Software Engineering*, 19(4):1144–1168, 2014.

[69] Jerónimo Irazábal, Claudia Pons, and Carlos Neil. Model transformation as a mechanism for the implementation of domain specific transformation languages. *SADIO Electronic Journal of Informatics and Operations Research*, 9(1), 2010.

[70] Xiaoping Jia, Hongming Liu, Lizhang Qin, and Adam Steele. Metamodel based Model Transformation Framework. In Hamid R. Arabnia and Hassan Reza, editors, *Software Engineering Research and Practice*, pages 496–502, Las Vegas, Nevada, USA, 2008. CSREA Press.

[71] Timo Jokela, Netta Iivari, Juha Matero, and Minna Karukka. The standard of user-centered design and the standard definition of usability: analyzing ISO 13407 against ISO 9241-11. In *Proceedings of the Latin American conference on Human-computer interaction*, pages 53–60. ACM, 2003.

[72] Albin Jossic, Marcos Didonet Del Fabro, Jean-Philippe 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. IEEE, 2007.

[73] Frédéric Jouault. QVT to ATLVM.
http : //www.eclipse.org/atl/usecases/QVT2ATLVM/.

[74] Frédéric Jouault. Loosely coupled traceability for atl. In *ECMDA Traceability Workshop (ECMDA-TW)*, pages 29–37. Citeseer, 2005.
http : //www.eclipse.org/atl/atlTransformations/#ATL2Tracer.

[75] Frédéric Jouault, Jean Bézivin, and Ivan Kurtev. TCS: a DSL for the specification of textual concrete syntaxes in model engineering. In *Proceedings of the 5th international conference on Generative programming and component engineering*, pages 249–254. ACM, 2006.

[76] Frédéric Jouault and Ivan Kurtev. Transforming models with ATL. In Heidelberg Berlin, editor, *Proceedings of the 2005 International Conference on Satellite Events at the MoDELS*, volume 3844 of *Lecture Notes in Computer Science*, pages 128–138, Montego Bay, Jamaica, 2006. Springer-Verlag.

[77] Audris Kalnins, Janis Barzdins, and Edgars Celms. Model Transformation Language MOLA. In Uwe Aßmann, Mehmet Aksit, and Arend Rensink, editors, *MDAFA*, volume 3599 of *Lecture Notes in Computer Science*, pages 62–76, Twente, The Netherlands, 2004. Springer.

[78] Pallavi Kalyanasundaram and Sunita P Ugale. Model Transformation: Concept, Current Trends and Challenges. *International Journal of Computer Applications*, 119(14), 2015.

[79] Gabor Karsai, Holger Krahn, Claas Pinkernell, Bernhard Rumpe, Martin Schindler, and Steven Völkel. Design guidelines for domain specific languages. *arXiv preprint arXiv:1409.2378*, 2014.

[80] Amogh Kavimandan, Reinhard Klemm, and Aniruddha Gokhale. Automated Context-Sensitive dialog synthesis for enterprise workflows using templatized model transformations. In *Enterprise Distributed Object Computing Conference, 2008. EDOC'08. 12th International IEEE*, pages 159–168. IEEE, 2008.

[81] Steven Kelly and Risto Pohjonen. Domain-specific modelling for cross-platform product families. In *International Conference on Conceptual Modeling*, pages 182–194. Springer, 2002.

[82] Steven Kelly and Juha-Pekka Tolvanen. *Domain-specific modeling: enabling full code generation*. John Wiley & Sons, 2008.

[83] Stuart Kent. Model Driven Engineering. In *Proceedings of the Third International Conference on Integrated Formal Methods (IFM'02)*, volume 2335 of *Lecture Notes in Computer Science*, pages 286–298, London, UK, 2002. Springer-Verlag.

[84] Benjamin Klatt. Xpand: A closer look at the model2text transformation language. *Language*, 10(16):2008, 2007.

[85] Anneke Kleppe. *Software language engineering: creating domain-specific languages using metamodels*. Pearson Education, 2008.

[86] Anneke G Kleppe, Jos Warmer, Wim Bast, and MDA Explained. The model driven architecture: practice and promise, 2003.

[87] Paul Klint, Ralf Lämmel, and Chris Verhoef. Toward an engineering discipline for grammarware. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 14(3):331–380, 2005.

[88] Philippe Kruchten. *The rational unified process: an introduction*. Addison-Wesley Professional, 2004.

[89] Marco Kuhrmann. You Can'T Tailor What You Haven'T Modeled. In *Proceedings of the 2014 International Conference on Software and System Process*, ICSSP 2014, pages 189–190, New York, NY, USA, 2014. ACM.

[90] Marco Kuhrmann, Daniel Méndez Fernández, and Ragna Steenweg. Systematic software process development: Where do we stand today? In *Proceedings of the 2013 International Conference on Software and System Process*, ICSSP 2013, pages 166–170, New York, NY, USA, 2013. ACM.

[91] Ivan Kurtev, Jean Bézivin, and Mehmet Akşit. Technological Spaces: An Initial Appraisal. In *International Conference on Cooperative Information Systems (CoopIS), DOA'2002 Federated Conferences, Industrial Track, Irvine, USA*, pages 1–6, October 2002.

[92] Ivan Kurtev, Jean Bézivin, Frédéric Jouault, and Patrick Valduriez. Model-based DSL frameworks. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, pages 602–616. ACM, 2006.

[93] Thomas K Landauer. *The trouble with computers: Usefulness, usability, and productivity*, volume 21. Taylor & Francis, 1995.

[94] Michael Lawley and Jim Steel. Practical Declarative Model Transformation with Tefkat. In Jean-Michel Bruel, editor, *MoDELS Satellite Events*, volume 3844 of *Lecture Notes in Computer Science*, pages 139–150, Montego Bay, Jamaica, 2005. Springer.

[95] Jacques Lonchamp. A Structured Conceptual and Terminological Framework for Software Process Engineering. In *In Proceedings of the Second International Conference on the Software Process*, pages 41–53. IEEE Computer Society Press, 1993.

[96] Miles Macleod, Rosemary Bowden, Nigel Bevan, and Ian Curson. The MUSiC performance measurement method. *Behaviour & Information Technology*, 16(4-5):279–293, 1997.

[97] Ivano Malavolta, Henry Muccini, Patrizio Pelliccione, and Damien Andrew Tamburri. Providing architectural languages and tools interoperability through model transformation technologies. *Software Engineering, IEEE Transactions on*, 36(1):119–140, 2010.

[98] Tom Mens and Pieter Van Gorp. A Taxonomy of Model Transformation. *Electr. Notes Theor. Comput. Sci.*, 152:125–142, 2006.

[99] Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and how to develop domain-specific languages. *ACM Computing Surveys (CSUR)*, 37(4):316–344, 2005.

[100] ModelPlex Project. Deliverable D2.1.a: "Global Model Management Principles", March 2008. $http://docatlanmod.emn.fr/AM3/Documentation/D2-1-a_Global_Model_Management_Principles_v1-1.pdf, Last visited December 2016.$

[101] Parastoo Mohagheghi, Vegard Dehlen, and Tor Neple. Definitions and approaches to model quality in model-based software development–A review of literature. *Information and Software Technology*, 51(12):1646–1669, 2009.

[102] Parastoo Mohagheghi, Miguel A Fernandez, Juan A Martell, Mathias Fritzsche, and Wasif Gilani. Mde adoption in industry: challenges and success criteria. In *International Conference on Model Driven Engineering Languages and Systems*, pages 54–59. Springer, 2008.

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

[104] Shiva Nejati, Mehrdad Sabetzadeh, Marsha Chechik, Steve Easterbrook, and Pamela Zave. Matching and merging of statecharts specifications. In *Software Engineering, 2007. ICSE 2007. 29th International Conference on*, pages 54–64. IEEE, 2007.

[105] Oscar Nierstrasz, Jon Whittle, and David Harel, editors. *Model Driven Engineering Languages and Systems, 9th International Conference, MoDELS 2006, Genova, Italy, October 1-6, 2006, Proceedings*, volume 4199 of *Lecture Notes in Computer Science*. Springer, 2006.

[106] Alexis Ocampo, Fabio Bella, and Jürgen Münch. Software process commonality analysis. *Software Process: Improvement and Practice*, 10(3):273–285, 2005.

[107] Jon Oldevik and Oystein Haugen. Higher-order transformations for product lines. In *null*, pages 243–254. IEEE, 2007.

[108] Jon Oldevik, Tor Neple, Roy Grønmo, Jan Aagedal, and Arne-J Berre. Toward standardised model-to-text transformations. In *Model Driven Architecture–Foundations and Applications*, pages 239–253. Springer, 2005.

[109] Nuno Oliveira, Maria Joao Varanda Pereira, Pedro R. Henriques, and Daniela da Cruz. Domain-specific languages: A theoretical survey. In *Proceedings of the 3rd Compilers, Programming Languages, Related Technologies and Applications (CoRTA'2009)*, pages

35–46, Lisbon, Portugal, 2009. Faculdade de Ciências da Universidade de Lisboa.

[110] Daniel Ortega, Luis Silvestre, María Cecilia Bastarrica, and Sergio F Ochoa. A tool for modeling software development contexts in small software organizations. In *SCCC 2012, Proceedings of the XXXI International Conference of the Chilean Computer Science Society, Valparaiso, Chile, November 2012*, pages 29–35. IEEE, 2012.

[111] Leon Osterweil. Software Processes Are Software Too. In *Proceedings of the 9th International Conference on Software Engineering*, ICSE '87, pages 2–13, Los Alamitos, CA, USA, 1987. IEEE Computer Society Press.

[112] Harkirat Padda. *QUIM: A Model for Usability/Quality in Use Measurement*. LAP Lambert Academic Publishing, Germany, 2009.

[113] Richard F. Paige, Jonathan S. Ostroff, and Phillip J Brooke. Principles for modeling language design. *Information and Software Technology*, 42(10):665–675, 2000.

[114] Mikael Peltier. MTrans, a DSL for model transformation. In *Enterprise Distributed Object Computing Conference, 2002. EDOC'02. Proceedings. Sixth International*, pages 190–199. IEEE, 2002.

[115] 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.

[116] Aleksandar Popovic, Ivan Lukovic, Vladimir Dimitrieski, and Verislav Djukic. A DSL for modeling application-specific functionalities of business applications. *Computer Languages, Systems & Structures*, 43:69–95, 2015.

[117] ADAPTE Fondef Project. Adaptable Domain and Process Technology Engineering, grant D09I1171. Fondef, Conicyt, Gobierno de Chile. http://www.adapte.cl, Last visited October 2016.

[118] AMW project. AMW Traceability. *http://www.eclipse.org/gmt/amw/usecases/traceability/*.

[119] ATL project. ATL to Binding Debugger. *http://www.eclipse.org/atl/atlTransformations/#ATL2BindingDebugger*.

[120] ATL project. ATL to BindingDebugger. *http://www.eclipse.org/atl/atlTransformations/#ModelMeasurement*.

[121] ATL project. ATL to Problem. *http://www.eclipse.org/atl/atlTransformations/#ATL2Problem*.

[122] ATL project. KM3 to ATL Copier. *http://www.eclipse.org/atl/atlTransformations/#KM32ATLCopier*.

[123] AWM project. AMW to ATL and XSLT.

http : *//www.eclipse.org/gmt/amw/examples/#AMW*$_2$*ATL*$_X$*SLT*.

[124] AWM project. Translating KM3 into SQL DDL using AMW and ATL. *https* : *//eclipse.org/gmt/amw/examples/#AMW*$_K$*M32SQL*.

[125] Alcides Quispe, Maira Marques, Luis Silvestre, Sergio F. Ochoa, and Romain Robbes. Requirements engineering practices in very small software enterprises: A diagnostic study. In Sergio F. Ochoa, Federico Meza, Domingo Mery, and Claudio Cubillos, editors, *SCCC 2010, Proceedings of the XXIX International Conference of the Chilean Computer Science Society, Antofagasta, Chile, November 2010*, pages 81–87. IEEE Computer Society, 2010.

[126] Jolita Ralyté, Rébecca Deneckère, and Colette Rolland. Towards a generic model for situational method engineering. In *Proceedings of the 15th International Conference on Advanced Information Systems Engineering*, CAiSE'03, pages 95–110, Berlin, Heidelberg, 2003. Springer-Verlag.

[127] Colin Robson and Kieran McCartan. *Real world research*. Wiley, 2016.

[128] Colette Rolland and Selmin Nurcan. Business Process Lines to deal with the Variability. In *System Sciences (HICSS), 2010 43rd Hawaii International Conference on*, pages 1–10. IEEE, 2010.

[129] Dieter Rombach. Integrated software process and product lines. In *Unifying the Software Process Spectrum*, pages 83–90. Springer, 2005.

[130] Louis M Rose, Richard F Paige, Dimitrios S Kolovos, and Fiona AC Polack. The epsilon generation language. In *European Conference on Model Driven Architecture-Foundations and Applications*, pages 1–16. Springer, 2008.

[131] Winston W Royce. Managing the development of large software systems. In *proceedings of IEEE WESCON*, volume 26, pages 1–9. Los Angeles, 1970.

[132] Bernhard Rumpe and Ingo Weisemöller. A Domain Specific Transformation Language. *In: ME 2011 - Models And Evolution, Welington, New Zealand*, October 2011.

[133] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, 2009.

[134] Jesús Sánchez-Cuadrado and Jesús García Molina. A Plugin-Based Language to Experiment with Model Transformation. In Nierstrasz et al. [105], pages 336–350.

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

[136] Ken Schwaber. Scrum development process. In *Business Object Design and Implementation*, pages 117–134. Springer, 1997.

[137] Richard W Selby. *Software engineering: Barry W. Boehm's lifetime contributions to*

software development, management, and research, volume 69. John Wiley & Sons, 2007.

[138] Bran Selic. The pragmatics of model-driven development. *IEEE software*, 20(5):19, 2003.

[139] Shane Sendall and Wojtek Kozaczynski. Model Transformation: The Heart and Soul of Model-Driven Software Development. *IEEE Software*, 20(5):42–45, 2003.

[140] Marten Sijtema. Introducing Variability Rules in ATL for Managing Variability in MDE-based Product Lines. volume 10, pages 39–49. Citeseer, 2010.

[141] Luis Silvestre. Automatic generation of transformations for software process tailoring. In Mira Balaban and Martin Gogolla, editors, *Proceedings of the ACM Student Research Competition at MODELS 2015 co-located with the ACM/IEEE 18th International Conference MODELS 2015, Ottawa, Canada, September 29, 2015.*, volume 1503 of *CEUR Workshop Proceedings*, pages 46–51. CEUR-WS.org, 2015.

[142] Luis Silvestre, María Cecilia Bastarrica, and Sergio F. Ochoa. HOTs for Generating Transformations with Two Input Models. In *SCCC 2013, Proceedings of the XXXII International Conference of the Chilean Computer Science Society, Temuco, Chile, November 2013*, pages 26–29, 2013.

[143] Luis Silvestre, María Cecilia Bastarrica, and Sergio F. Ochoa. A Model-based Tool for Generating Software Process Model Tailoring Transformations. In Luís Ferreira Pires, Slimane Hammoudi, and Joaquim Filipe, editors, *MODELSWARD*, pages 533–540. SciTePress, 2014.

[144] Luis Silvestre, María Cecilia Bastarrica, and Sergio F. Ochoa. A usable mde-based tool for software process tailoring. In Vinay Kulkarni and Omar Badreddin, editors, *Proceedings of the MoDELS 2015 Demo and Poster Session co-located with ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2015), Ottawa, Canada, September 27, 2015.*, volume 1554 of *CEUR Workshop Proceedings*, pages 36–39. CEUR-WS.org, 2015.

[145] Borislava I Simidchieva, Lori A Clarke, and Leon J Osterweil. Representing process variation with a process family. In *International Conference on Software Process*, pages 109–120. Springer, 2007.

[146] Jocelyn Simmonds, Daniel Perovich, María Cecilia Bastarrica, and Luis Silvestre. A megamodel for Software Process Line modeling and evolution. In Timothy Lethbridge, Jordi Cabot, and Alexander Egyed, editors, *18th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MoDELS 2015, Ottawa, ON, Canada, September 30 - October 2, 2015*, pages 406–415. IEEE, 2015.

[147] Dag IK Sjøberg, Jo E Hannay, Ove Hansen, Vigdis By Kampenes, Amela Karahasanovic, Nils-Kristian Liborg, and Anette C Rekdal. A survey of controlled experiments in software engineering. *Software Engineering, IEEE Transactions on*, 31(9):733–753, 2005.

[148] Ian Sommerville. Software engineering. international computer science series. *ed: Addison Wesley*, 2004.

[149] Jean-Sébastien Sottet, Vincent Ganneau, Gaëlle Calvary, Joëlle Coutaz, Alexandre Demeure, Jean-Marie Favre, and Rachel Demumieux. Model-driven adaptation for plastic user interfaces. In *Human-Computer Interaction–INTERACT 2007*, pages 397–410. Springer, 2007.

[150] Indian Standard. Systems and software engineering–software life cycle processes. *ISO Standard*, 12207:2008, 2008.

[151] Yu Sun, Jules White, and Jeff Gray. Model Transformation by Demonstration. In Andy Schürr and Bran Selic, editors, *MoDELS*, volume 5795 of *Lecture Notes in Computer Science*, pages 712–726, Denver, CO, USA,, 2009. Springer.

[152] Eugene Syriani, Jörg Kienzle, and Hans Vangheluwe. Exceptional transformations. In *International Conference on Theory and Practice of Model Transformations*, pages 199–214. Springer, 2010.

[153] Eugene Syriani, Hans Vangheluwe, Raphael Mannadiar, Conner Hansen, Simon Van Mierlo, and Hüseyin Ergin. AToMPM: A Web-based Modeling Environment. In *Demos/Posters/StudentResearch@ MoDELS*, pages 21–25. Citeseer, 2013.

[154] CMMI Product Team. CMMI for Development, Version 1.3. Technical Report CMU/SEI-2010-TR-033, Software Engineering Institute, 2010.

[155] Massimo Tisi, Jordi Cabot, and Frédéric Jouault. Improving Higher-Order Transformations Support in ATL. In Laurence Tratt and Martin Gogolla, editors, *ICMT*, volume 6142 of *Lecture Notes in Computer Science*, pages 215–229, Malaga, Spain, 2010. Springer.

[156] Massimo Tisi, Frédéric Jouault, Piero Fraternali, Stefano Ceri, and Jean Bézivin. On the Use of Higher-Order Model Transformations. In Richard F. Paige, Alan Hartman, and Arend Rensink, editors, *ECMDA-FA*, volume 5562 of *Lecture Notes in Computer Science*, pages 18–33, Enschede, The Netherlands, 2009. Springer.

[157] Claudia Valtierra, Mirna Muñoz, and Jezreel Mejia. Characterization of software processes improvement needs in SMEs. In *Mechatronics, Electronics and Automotive Engineering (ICMEAE), 2013 International Conference on*, pages 223–228. IEEE, 2013.

[158] Marcel F. van Amstel, Christian F.J. Lange, and Mark G.J. van den Brand. Using metrics for assessing the quality of ASF+ SDF model transformations. In *International Conference on Theory and Practice of Model Transformations*, pages 239–248. Springer, 2009.

[159] Arie van Deursen, Paul Klint, and Joost Visser. Domain-Specific Languages: An Annotated Bibliography. *SIGPLAN Notices*, 35(6):26–36, 2000.

[160] Simon Van Mierlo. Higher-Order Transformations in AToMPM. Technical Report

MSDL-HOTS, Modelling, Simulation and Design Lab, 2012.
http : *//msdl.cs.mcgill.ca/people/simonvm/files/hots*.

[161] Dániel Varró and András Pataricza. Generic and meta-transformations for model transformation engineering. In *«UML» 2004—The Unified Modeling Language. Modeling Languages and Applications*, pages 290–304. Springer, 2004.

[162] Dániel Varró, Gergely Varró, and András Pataricza. Designing the automatic transformation of visual languages. *Science of Computer Programming*, 44(2):205–227, 2002.

[163] Dániel Varró. Model Transformation by Example. In Oscar Nierstrasz, Jon Whittle, David Harel, and Gianna Reggio, editors, *Model Driven Engineering Languages and Systems*, volume 4199 of *Lecture Notes in Computer Science*, pages 410–424. Springer Berlin Heidelberg, 2006.

[164] Markus Voelter, Sebastian Benz, Christian Dietrich, Birgit Engelmann, Mats Helander, Lennart CL Kats, Eelco Visser, and Guido Wachsmuth. *DSL engineering: Designing, implementing and using domain-specific languages*. dslbook. org, 2013.

[165] Markus Völter, Thomas Stahl, Jorn Bettin, Arno Haase, and Simon Helsen. *Model-driven software development: technology, engineering, management*. John Wiley & Sons, 2013.

[166] Dennis Wagelaar. Composition techniques for rule-based model transformation languages. In *Theory and Practice of Model Transformations*, pages 152–167. Springer, 2008.

[167] Hironori Washizaki. Building software process line architectures from bottom up. In *International Conference on Product Focused Software Process Improvement*, pages 415–421. Springer, 2006.

[168] WebRatio. WebRatio Web Platform.
http : *//www.webratio.com/site/content/en/web* − *application* − *development*.

[169] David Weiss, J.J. Li, H. Slye, T. Dinh-Trong, and Sun Hongyu. Decision-Model-Based Code Generation for SPLE. In *SPLC*, pages 129–138, Sept 2008.

[170] Jon Whittle, John Hutchinson, and Mark Rouncefield. The state of practice in model-driven engineering. *Software, IEEE*, 31(3):79–85, 2014.

[171] Manuel Wimmer and Gerhard Kramler. Bridging grammarware and modelware. In *Satellite Events at the MoDELS 2005 Conference*, pages 159–168. Springer, 2005.

[172] Andrés Yie, Rubby Casallas, Dirk Deridder, and Dennis Wagelaar. Realizing model transformation chain interoperability. *Software & Systems Modeling*, 11(1):55–75, 2012.

[173] Robert K. Yin. *Case study research: design and methods*. Sage Publications, Newbury Park, CA, third edition, 2002.

[174] Sami Zahran. *Software process improvement: practical guidelines for business susccess*. Addison-Wesley Longman Ltd., 1998.

Annex A

Listings of the Higher-order Transformation

A.1 HOT code for generating the header

Listing A.1: Excerpt of the HOT for Generating the Header

```
module VDM2ATL;
create OUT : ATLMeta from IN : VDMM;

rule Module {
from
vdm : VDMM!DecisionModel
to
atl : ATLMeta!Module (
isRefining <- false ,
name <- 'TailoringTransformation',
outModels <- outputModels_spem,
inModels <- inputModels_spem,
inModels <- inputModels_spcm
),
outputModels_spem:ATLMeta!OclModel(
name <- 'OUT',
metamodel<- outputMetamodel_spem
),
outputMetamodel_spem:ATLMeta!OclModel(
name <- 'MM2'
),
inputModels_spem: ATLMeta!OclModel(
name <- 'IN',
metamodel <-inputMetamodel_spem
),
inputMetamodel_spem:ATLMeta!OclModel(
name <- 'MM'
),
inputModels_spcm: ATLMeta!OclModel(
name <- 'IN1',
metamodel <-inputMetamodel_spcm
),
inputMetamodel_spcm:ATLMeta!OclModel(
name <- 'MM1'
)
do{
thisModule.helpergetContextAttributeConfiguration(vdm);
thisModule.helpergetTaskDefinition(vdm);
thisModule.helperOptionalRules(vdm);
thisModule.createMethodLibrary(vdm);
thisModule.createMethodPlugin(vdm);
}
}
```

A.2 HOT code for generating the matched rules

Listing A.2: Excerpt of the HOT for Generating Matched Rules

```
rule createMethodPlugin(vdm : VDM!DecisionModel){
to
-- InPattern y outPattern
methodPluginRule : ATLMeta!MatchedRule(
name <- 'methodPlugin',
isAbstract <- false ,
isRefining <- false ,
module <- thisModule.resolveTemp(vdm, 'atl'),
inPattern <- methodPluginInPattern,
outPattern <- methodPluginOutPattern
),
-- From Section
methodPluginInPattern : ATLMeta!InPattern(
elements <- Set{methodPluginInElement}
),
methodPluginInElement : ATLMeta!SimpleInPatternElement(
varName <- 'mp',
type <- inOclType
),
inOclType : ATLMeta!OclModelElement(
name <- 'MethodPlugin',
model <- thisModule.resolveTemp(vdm, 'inputMetamodel_spem')
),
-- To Section
methodPluginOutPattern : ATLMeta!OutPattern(
elements <- Set{methodPluginOutElement}
),
methodPluginOutElement : ATLMeta!SimpleOutPatternElement(
varName <- 'mpp',
type <- outOclType,
bindings <- bindingsMethodPluginOutElement_name,
bindings <- bindingsMethodPluginOutElemen_description,
bindings <- bindingsMethodPluginOutElemen_ownedProcessPackage,
bindings <- bindingsMethodPluginOutElemen_ownedMethodContentPackage
),
outOclType : ATLMeta!OclModelElement(
name <- 'MethodPlugin',
model <- thisModule.resolveTemp(vdm, 'outputMetamodel_spem')
),
bindingsMethodPluginOutElement_name:ATLMeta!Binding(
propertyName <-'name',
value <-bindingsMethodPluginOutElement_nameValue
),
bindingsMethodPluginOutElement_nameValue:ATLMeta!NavigationOrAttributeCallExp(
name <-'name',
source<-variableExp_bindingsMethodPluginOutElement_nameValueSource
),
variableExp_bindingsMethodPluginOutElement_nameValueSource:ATLMeta!VariableExp(
referredVariable<-methodPluginInElement
),

bindingsMethodPluginOutElemen_description:ATLMeta!Binding(
propertyName <-'description',
value <-bindingsMethodPluginOutElemen_descriptionValue
),
bindingsMethodPluginOutElemen_descriptionValue:ATLMeta!NavigationOrAttributeCallExp(
name <-'description',
source<-bindingsMethodPluginOutElemen_descriptionValueSource
),
bindingsMethodPluginOutElemen_descriptionValueSource:ATLMeta!VariableExp(
referredVariable<-methodPluginInElement
),

bindingsMethodPluginOutElemen_ownedProcessPackage:ATLMeta!Binding(
propertyName <-'ownedProcessPackage',
value <-bindingsMethodPluginOutElemen_ownedProcessPackageValue
),
bindingsMethodPluginOutElemen_ownedProcessPackageValue:ATLMeta!NavigationOrAttributeCallExp(
name <-'ownedProcessPackage',
source<-bindingsMethodPluginOutElemen_ownedProcessPackageValueSource
),
bindingsMethodPluginOutElemen_ownedProcessPackageValueSource:ATLMeta!VariableExp(
referredVariable<-methodPluginInElement
),

bindingsMethodPluginOutElemen_ownedMethodContentPackage:ATLMeta!Binding(
propertyName <-'ownedMethodContentPackage',
value <-bindingsMethodPluginOutElemen_ownedMethodContentPackageValue
),
bindingsMethodPluginOutElemen_ownedMethodContentPackageValue:ATLMeta!NavigationOrAttributeCallExp(
name <-'ownedMethodContentPackage',
source<-bindingsMethodPluginOutElemen_ownedMethodContentPackageValueSource
),
bindingsMethodPluginOutElemen_ownedMethodContentPackageValueSource:ATLMeta!VariableExp(
referredVariable<-methodPluginInElement
)    }
```

A.3 HOT code for generating the helpers

Listing A.3: Excerpt of the HOT for Generating Helpers

```
rule helpergetContextAttributeConfiguration(vdm : VDMM!DecisionModel){
to
getContextAttributeHelper: ATLMeta!Helper(
module <- thisModule.resolveTemp(vdm, 'atl'),
definition<-definitionHelper
),
definitionHelper:ATLMeta!OclFeatureDefinition(
feature <- featureOclFeatureDefinition

),

featureOclFeatureDefinition: ATLMeta!Operation(
name <- 'getContextAttributeConfiguration',
parameters<-parameterOperation,
returnType<-returnTypeOperation,
body<-bodyOperation
),

parameterOperation: ATLMeta!Parameter(
varName <- 'nameAttribute',
type<-typeParameter
),

returnTypeOperation:ATLMeta!OclModelElement(
name<-'ContextAttributeConfiguration',
model<-thisModule.resolveTemp(vdm, 'inputMetamodel_spcm')
),

typeParameter:ATLMeta!StringType(
),

bodyOperation:ATLMeta!CollectionOperationCallExp(
operationName <-'first',
source<-sourceCollectionOperation
),

sourceCollectionOperation:ATLMeta!IteratorExp(
name<-'select',
source<-sourceIteratorExp,
body<-bodyCollectionOperationCall,
iterators<-iteratorsIterator
),

iteratorsIterator:ATLMeta!Iterator(
varName<-'a',
variableExp<-sourceVariableExp
),

sourceIteratorExp:ATLMeta!CollectionOperationCallExp(
operationName<-'asSequence',
source<-sourceCollectionOperationCall
),

sourceCollectionOperationCall:ATLMeta!OperationCallExp(
operationName<-'allInstances',
source<-sourceOperationCallEx
),

sourceOperationCallEx:ATLMeta!OclModelElement(
name<-'ContextAttributeConfiguration',
model<-thisModule.resolveTemp(vdm, 'inputMetamodel_spcm')
),

bodyCollectionOperationCall:ATLMeta!OperatorCallExp(
operationName<-'=',
source<-sourceNavigationOrAttributeCallExp,
arguments<-argumentsVariableExp
),

sourceNavigationOrAttributeCallExp:ATLMeta!NavigationOrAttributeCallExp(
name<-'name',
source<-sourceNavigationOrAttributeCallExp2
),
sourceNavigationOrAttributeCallExp2:ATLMeta!NavigationOrAttributeCallExp(
name<-'myContextElement',
source<-sourceVariableExp
),
sourceVariableExp:ATLMeta!VariableExp(
referredVariable<-iteratorsIterator
),
argumentsVariableExp:ATLMeta!VariableExp(
referredVariable<-parameterOperation
)
}
```

A.4 Tailoring transformation generated by the HOT and the ATL extractor

Listing A.4: Tailoring transformation in ATL that is generated by the HOT and the ATL extractor

```
module Tailoring;

create OUT : MM2 from IN : MM, IN1 : MM1;

helper def: getContextAttributeConfiguration (nameAttribute: String) :
MM1!ContextAttributeConfiguration = MM1!ContextAttributeConfiguration.allInstances()->asSequence()->select(
    a | a.myContextElement.name =nameAttribute )->first();
helper def: getValue(nameAttribute: String):
String = thisModule.getContextAttributeConfiguration(nameAttribute).myContextAttributeValue.value;

helper def:getTaskDefinition(taskDefinitionName:String): MM!TaskDefinition =
MM!TaskDefinition.allInstances()->asSequence()->select(t|t.name = taskDefinitionName)->first();


helper def:nextElement(a:MM!WorkBreakDownElement): MM!WorkBreakDownElement = MM!WorkBreakDownElement.
    allInstances()->select(t|t=a.next)->first();
helper def:next(a:MM!WorkBreakDownElement): MM!WorkBreakDownElement = if(thisModule.optionalRule(thisModule
    .nextElement(a).name)) then a else thisModule.next(thisModule.nextElement(a)) endif;

helper def: optionalRule(name:String): Boolean =
if(Sequence{'Requirements','Design') then(
if('Design'= name) then
thisModule.ruleOpt2()
else (
if('Requirements'= name) then
thisModule.ruleOpt1()
else true
endif
) endif
) else true endif;

helper def: alternativeRule(tu:MM!TaskUse): MM!TaskDefinition =
if(Sequence{'Specify Requirements', 'Establish Requirements Baseline'}.includes(tu.name)) then(
if('Establish Requirements Baseline'= tu.name) then
thisModule.ruleAlt2(tu)
else (
if('Specify Requirements'= tu.name) then
thisModule.ruleAlt1(tu)
else tu.linkTask
endif
) endif
) else tu.linkTask endif;

helper def:ruleOpt1():Boolean=if(thisModule.getValue('Project Type') = 'Maintenance-adaptation' and
    thisModule.getValue('Project Duration') = 'Small') then false
else true
endif;

helper def:ruleOpt2():Boolean=if(thisModule.getValue('Project Type') = 'Maintenance-correction' and
    thisModule.getValue('Project Duration') = 'Small') then false
else true
endif;

helper def:ruleAlt1(tu:MM!TaskUse): MM!TaskDefinition=if((thisModule.getValue('Project Type') = 'Incidents'
    or thisModule.getValue('Project Type') = 'Maintenance-enhancement') and thisModule.getValue('Business
    Knowledge') = 'Known') then thisModule.getTaskDefinition('Specify Requirements in plain text')
else thisModule.getTaskDefinition(tu.name)
endif;

helper def:ruleAlt2(tu:MM!TaskUse): MM!TaskDefinition=if(thisModule.getValue('Project Type') = 'New
    development' and thisModule.getValue('Project Duration') = 'Medium' and thisModule.getValue('Business
    Knowledge') = 'Unknown') then thisModule.getTaskDefinition('Establish Requirements Baseline and Test
    Cases')
else thisModule.getTaskDefinition(tu.name)
endif;

rule main{
from     ml:MM!MethodLibrary
to               mll:MM2!MethodLibrary(
name <- ml.name,
description <- ml.description,
ownedMethodPlugin<- ml.ownedMethodPlugin,
predefinedConfiguration<-ml.predefinedConfiguration
)
}
rule methodplugin{
from     mp:MM!MethodPlugin
to               mpp:MM2!MethodPlugin (
name<- mp.name,
description <- mp.description,
ownedProcessPackage<- mp.ownedProcessPackage,
ownedMethodContentPackage <- mp.ownedMethodContentPackage
)
```

```
74 }
75 rule methodconfiguration{
76 from c:MM!MethodConfiguration
77 to       cc:MM2!MethodConfiguration(
78 name <-c .name,
79 description <- c.description,
80 baseConfiguration<-c.baseConfiguration,
81 methodPluginSelection<-c.methodPluginSelection,
82 defaultView<-c.defaultView,
83 processView<-c.processView,
84 processPackageSelection<-c.processPackageSelection,
85 myProcessPackage<-c.myProcessPackage,
86 contentSelection<-c.contentSelection
87 )
88 }
89 rule ProcessPackage{
90 from pp:MM!ProcessPackage
91 to ppp:MM2!ProcessPackage
92 (
93 name <- pp.name,
94 processElements <- pp.processElements,
95 processPackages <- pp.processPackages
96 )
97 }
98 rule Activity{
99 from a:MM!Activity(
100 thisModule.optionalRule(a.name)
101 )
102 to aa:MM2!Activity(
103 name <- a.name,
104 nestedElements <- a.nestedElements,
105 processPerformer <- a.processPerformer,
106 processParameter <- a.processParameter,
107 usedActivity <- a.usedActivity,
108 useKind <- a.useKind,
109 next <- a.next,
110 ownedParemeter <- a.ownedParemeter,
111 description <- a.description,
112 variabilityType <- a.variabilityType,
113 variabilityBasedOnElement <- a.variabilityBasedOnElement)
114 }
115 rule TaskUse{
116 from tu:MM!TaskUse(thisModule.optionalRule(tu.name))
117
118 using{
119 task:MM!TaskDefinition = thisModule.alternativeRule(tu);
120 }
121
122 to tuu:MM2!TaskUse(
123 name <- tu.name,
124 linkTask<- task,
125 next <- tu.next,
126 ownedParemeter <- tu.ownedParemeter,
127 description <- tu.description)
128 }
129 rule RoleUse{
130 from ru:MM!RoleUse
131 to ruu:MM2!RoleUse(
132 name <- ru.name,
133 linkRole<-ru.linkRole,
134 myPerformer <- ru.myPerformer,
135 ownedParemeter <- ru.ownedParemeter,
136 description <- ru.description
137 )
138 }
139 rule WorkProductUse{
140 from wpu:MM!WorkProductUse
141 to wpuu:MM2!WorkProductUse(
142 name <- wpu.name,
143 linkWorkProduct<-wpu.linkWorkProduct,
144 linkedProcessElement <- wpu.linkedProcessElement,
145 ownedParemeter <- wpu.ownedParemeter,
146 description <- wpu.description
147 )
148 }
149 rule MethodContentPackage{
150 from pp:MM!MethodContentPackage
151 to ppp:MM2!MethodContentPackage(
152 name <- pp.name,
153 methodContentElement <- pp.methodContentElement,
154 methodPackages <- pp.methodPackages,
155 categories <- pp.categories
156 )
157 }
158 rule Category{
159 from c:MM!Category
160 to cc:MM2!Category(
161 name<-c.name,
162 description <- c.description,
163 subCategory<- c.subCategory,
164 subCategories<- c.subCategories,
165 categorizedElement <- c.categorizedElement)
166 }
167 rule TaskDefinition{
168 from tu:MM!TaskDefinition
169 to tuu:MM2!TaskDefinition(
```

```
170 name <- tu.name,
171 description <- tu.description,
172 performer<-tu.performer,
173 participant<-tu.participant,
174 inputs<-tu.inputs,
175 outputs<-tu.outputs,
176 optionalInputs <- tu.optionalInputs,
177 variabilityBasedOnElement<-tu.variabilityBasedOnElement,
178 variabilityType<-tu.variabilityType)
179 }
180 rule RoleDefinition{
181 from ru:MM!RoleDefinition
182 to ruu:MM2!RoleDefinition(
183 description <- ru.description,
184 name <- ru.name,
185 variabilityType <- ru.variabilityType,
186 variabilityBasedOnElement <- ru.variabilityBasedOnElement)
187 }
188 rule WorkProductDefinition{
189 from wpu:MM!WorkProductDefinition
190 to wpuu:MM2!WorkProductDefinition(
191 description <- wpu.description,
192 name <- wpu.name,
193 variabilityType <- wpu.variabilityType,
194 variabilityBasedOnElement <- wpu.variabilityBasedOnElement)
195 }
```

Annex B

Evaluation Forms for validating ATAGeTT

B.1 Evaluation of organizational context definition

B.1.1 Usability Evaluation

Navigability					
	strongly disagree	disagree	neither agree nor disagree	agree	strongly agree
The navigation clearly shows the main frame of the navigation tool	☐	☐	☐	☐	☐
The graphical interface contains links that facilitate the navigation tool.	☐	☐	☐	☐	☐

Familiarity					
	strongly disagree	disagree	neither agree nor disagree	agree	strongly agree
The organizational context elements are understandable.	☐	☐	☐	☐	☐

The domain concepts have known meaning.	☐	☐	☐	☐	☐
The icons have known meaning.	☐	☐	☐	☐	☐

Flexibility					
	strongly disagree	disagree	neither agree nor disagree	agree	strongly agree
The tool-specific features are easy access.	☐	☐	☐	☐	☐
The tool allows users to create an organizational context definition (name, description, dimensions, attributes, values).	☐	☐	☐	☐	☐
The tool allows users to modify an organizational context definition (name, description, dimensions, attributes, values).	☐	☐	☐	☐	☐
The tool allows users to remove an organizational context definition.	☐	☐	☐	☐	☐
The tool supports other languages as English and Spanish.	☐	☐	☐	☐	☐

Simplicity					
	strongly disagree	disagree	neither agree nor disagree	agree	strongly agree
The information displayed in the tool is relevant for defining an organizational context .	☐	☐	☐	☐	☐
The information of the tool is sufficient for understanding the domain concepts.	☐	☐	☐	☐	☐
The tool content is classified.	☐	☐	☐	☐	☐

The tool content is correctly organized.	☐	☐	☐	☐	☐
The graphical interface is well distributed on the tool.	☐	☐	☐	☐	☐

Usefulness					
	strongly disagree	disagree	neither agree nor disagree	agree	strongly agree
The procedure to define organizational contexts is understandable.	☐	☐	☐	☐	☐
The procedure to define organizational contexts is clear.	☐	☐	☐	☐	☐
The interface of the tool is intuitive.	☐	☐	☐	☐	☐
The tool is appropriate to define organizational contexts.	☐	☐	☐	☐	☐
The tool is useful for defining organizational contexts.	☐	☐	☐	☐	☐
The tool is easy to use.	☐	☐	☐	☐	☐

B.1.2 Operability Evaluation

Organizational context definition					
	strongly disagree	disagree	neither agree nor disagree	agree	strongly agree
The process of the organizational context definition is clear.	☐	☐	☐	☐	☐
The process of the organizational context definition is flexible.	☐	☐	☐	☐	☐

The process of the organizational context definition is simple.	☐	☐	☐	☐	☐
The process of the organizational context definition is familiar.	☐	☐	☐	☐	☐

B.1.3 Software Quality Factors Evaluation

Organizational context definition					
	strongly disagree	disagree	neither agree nor disagree	agree	strongly agree
The domain concepts for defining the organizational context are simple to understand.	☐	☐	☐	☐	☐
The tool allows to build a required organizational context.	☐	☐	☐	☐	☐
The tool requires little training time to be used.	☐	☐	☐	☐	☐
The functions of the tool are well defined (edit, save).	☐	☐	☐	☐	☐

B.1.4 Questions

Do you consider that the tool is useful for your software company?

What would you suggest to improve the user experience when using the tool?

B.2 Evaluation of tailoring rules definition

B.2.1 Usability Evaluation

Navigability					
	strongly disagree	disagree	neither agree nor disagree	agree	strongly agree
The graphical interface clearly shows the main frame of the navigation tool.	☐	☐	☐	☐	☐
The graphical interface contains links that facilitate the navigation tool.	☐	☐	☐	☐	☐

Familiarity					
	strongly disagree	disagree	neither agree nor disagree	agree	strongly agree
The tailoring rules elements are understandable.	☐	☐	☐	☐	☐
The domain concepts have known meaning.	☐	☐	☐	☐	☐
The icons have known meaning.	☐	☐	☐	☐	☐

Consistency					
	strongly disagree	disagree	neither agree nor disagree	agree	strongly agree
The link name and action name are consistent.	☐	☐	☐	☐	☐
The title name and graphical content are consistent.	☐	☐	☐	☐	☐
The procedure to define tailoring rules is systematic.	☐	☐	☐	☐	☐

The tool fulfills its purpose, i.e. tailoring rules definition.	☐	☐	☐	☐	☐

Flexibility					
	strongly disagree	disagree	neither agree nor disagree	agree	strongly agree
The tool-specific features are easy access.	☐	☐	☐	☐	☐
The tool allows users to create a tailoring rule definition (variable element of process, condition, logical operator, conclusion).	☐	☐	☐	☐	☐
The tool allows users to modify a tailoring rule definition (context attribute and its respective value).	☐	☐	☐	☐	☐
The tool supports other languages as English and Spanish.	☐	☐	☐	☐	☐

Simplicity					
	strongly disagree	disagree	neither agree nor disagree	agree	strongly agree
The messages displayed in the tool are relevant for preventing possible errors.	☐	☐	☐	☐	☐
The information of the tool allows users to prevent possible errors.	☐	☐	☐	☐	☐
The tool does not induce to make mistakes.	☐	☐	☐	☐	☐

Facility					
	strongly disagree	disagree	neither agree nor disagree	agree	strongly agree
The information displayed in the tool is easy to understand.	☐	☐	☐	☐	☐
The information displayed in the tool is easy to identify.	☐	☐	☐	☐	☐
The tool content is correctly classified .	☐	☐	☐	☐	☐
The tool content is correctly organized.	☐	☐	☐	☐	☐
The tool content is well distributed.	☐	☐	☐	☐	☐

Usefulness					
	strongly disagree	disagree	neither agree nor disagree	agree	strongly agree
The procedure to define tailoring rules is understandable.	☐	☐	☐	☐	☐
The procedure to define tailoring rules is clear.	☐	☐	☐	☐	☐
The interface of the tool is intuitive.	☐	☐	☐	☐	☐
The tool is appropriate for defining rules.	☐	☐	☐	☐	☐
The tool is useful for defining rules.	☐	☐	☐	☐	☐
The tool is easy to use.	☐	☐	☐	☐	☐
The tool requires little user training time.	☐	☐	☐	☐	☐

Expressiveness					
	strongly disagree	disagree	neither agree nor disagree	agree	strongly agree
The tool allows users to generate the expected rules.	☐	☐	☐	☐	☐
The rules can be defined using the tool.	☐	☐	☐	☐	☐
The rules can be defined using conditions and conclusions.	☐	☐	☐	☐	☐
The rules can be in a correct manner.	☐	☐	☐	☐	☐
The tool allow user to define rules using an ideal mechanism.	☐	☐	☐	☐	☐

Support and documentation					
	strongly disagree	disagree	neither agree nor disagree	agree	strongly agree
The tool has help icons that guiding particular actions.	☐	☐	☐	☐	☐
The tool has specific help descriptions.	☐	☐	☐	☐	☐
The tool has help resources that are affordable.	☐	☐	☐	☐	☐

B.2.2 Expressiveness Evaluation

Tailoring Rules definition					
	strongly disagree	disagree	neither agree nor disagree	agree	strongly agree
Visibility of the tool.	☐	☐	☐	☐	☐
Understability of the domain concepts.	☐	☐	☐	☐	☐

Compatibility of the language.	☐	☐	☐	☐	☐
Orthogonality of the language.	☐	☐	☐	☐	☐
Orthogonality of the language.	☐	☐	☐	☐	☐
Robustness of the tool.	☐	☐	☐	☐	☐
Simplicity of the tool.	☐	☐	☐	☐	☐
Usefulness of the language.	☐	☐	☐	☐	☐
Suitability of the language.	☐	☐	☐	☐	☐
Support of the tool.	☐	☐	☐	☐	☐

B.2.3 Software Quality Factors Evaluation

Tailoring rules definition					
	strongly disagree	disagree	neither agree nor disagree	agree	strongly agree
The process of the tailoring rules definition is clear.	☐	☐	☐	☐	☐
The process of the tailoring rules definition is flexible.	☐	☐	☐	☐	☐
The process of the tailoring rules definition is simple.	☐	☐	☐	☐	☐
The process of the tailoring rules definition is familiar.	☐	☐	☐	☐	☐

B.2.4 Questions

Do you consider the tool useful for supporting the tailoring rules definition?

What would you suggest for improving the user experience while using the tool for tailoring rules definition?

B.3 Evaluation of project context definition

B.3.1 Usability Evaluation

Navigability					
	strongly disagree	disagree	neither agree nor disagree	agree	strongly agree
The graphical interface clearly shows the main frame of the navigation tool.	☐	☐	☐	☐	☐
The graphical interface contains links that facilitate the navigation tool.	☐	☐	☐	☐	☐

Familiarity					
	strongly disagree	disagree	neither agree nor disagree	agree	strongly agree
The project context elements are understandable.	☐	☐	☐	☐	☐
The domain concepts have known meaning.	☐	☐	☐	☐	☐
The icons have known meaning.	☐	☐	☐	☐	☐

Consistency					
	strongly disagree	disagree	neither agree nor disagree	agree	strongly agree
The link name and action name are consistent.	☐	☐	☐	☐	☐
The title name and graphical content are consistent.	☐	☐	☐	☐	☐

The procedure to define project contexts is systematic.	☐	☐	☐	☐	☐
The tool fulfills its purpose, i.e. software process tailoring.	☐	☐	☐	☐	☐

Flexibility					
	strongly disagree	disagree	neither agree nor disagree	agree	strongly agree
The tool-specific features are easy access.	☐	☐	☐	☐	☐
The tool allows users to create a project context definition (name, description, context, context attributes).	☐	☐	☐	☐	☐
The tool allows users to modify a project context definition (name, description, context, context attributes).	☐	☐	☐	☐	☐
The tool supports other languages and it is easy to switch between them.	☐	☐	☐	☐	☐

Simplicity					
	strongly disagree	disagree	neither agree nor disagree	agree	strongly agree
The information displayed in the tool is relevant for defining a project context.	☐	☐	☐	☐	☐
The information of the tool is sufficient for understanding the domain concepts.	☐	☐	☐	☐	☐
The tool content is classified.	☐	☐	☐	☐	☐
The tool content is correctly organized.	☐	☐	☐	☐	☐

The graphical interface content is well distributed on the tool.	☐	☐	☐	☐	☐

Usefulness					
	strongly disagree	disagree	neither agree nor disagree	agree	strongly agree
The procedure to define project contexts is understandable.	☐	☐	☐	☐	☐
The procedure to define project contexts is clear.	☐	☐	☐	☐	☐
The interface of the tool is intuitive.	☐	☐	☐	☐	☐
The tool is appropriate to define project contexts.	☐	☐	☐	☐	☐
The tool is easy to use.	☐	☐	☐	☐	☐

B.3.2 Operability Evaluation

Project context definition					
	strongly disagree	disagree	neither agree nor disagree	agree	strongly agree
The process of the software process tailoring is flexible.	☐	☐	☐	☐	☐
The process of the software process tailoring is useful.	☐	☐	☐	☐	☐
The process of the software process tailoring is correct.	☐	☐	☐	☐	☐
The process of the software process tailoring is simple.	☐	☐	☐	☐	☐

B.3.3 Software Quality Factors Evaluation

Project context definition					
	strongly disagree	disagree	neither agree nor disagree	agree	strongly agree
The concepts for defining the project context are simple to understand.	☐	☐	☐	☐	☐
The tool allows to build a required project context.	☐	☐	☐	☐	☐
The tool requires little training time to be used.	☐	☐	☐	☐	☐
The functions of the tool are well defined (edit, save).	☐	☐	☐	☐	☐

B.3.4 Questions

Do you consider the tool useful for supporting the project context definition?

Are you willing to adopt the automatic software process tailoring based on project context and tailoring rules? Why?