UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Department of Information Systems and Computation

Master in Software Engineering, Formal Methods and Information Systems

Master thesis

OpenUP/MDRE: A Model-Driven Requirements Engineering Approach for Health-Care Systems

Grzegorz Loniewski

Supervisor: prof. Emilio Insfran

Valencia 2010

This document was created in system LaTeX.

Contents

Abstract

The domains and problems for which it would be desirable to introduce information systems are currently very complex and the software development process is thus of the same complexity. One of these domains is health-care. Model-Driven Development (MDD) and Service-Oriented Architecture (SOA) are software development approaches that raise to deal with complexity, to reduce time and cost of development, augmenting flexibility and interoperability. However, many techniques and approaches that have been introduced are of little use when not provided under a formalized and well-documented methodological umbrella. A methodology gives the process a well-defined structure that helps in fast and efficient analysis and design, trouble-free implementation, and finally results in the software product improved quality.

While MDD and SOA are gaining their momentum toward the adoption in the software industry, there is one critical issue yet to be addressed before its power is fully realized. It is beyond dispute that requirements engineering (RE) has become a critical task within the software development process. Errors made during this process may have negative effects on subsequent development steps, and on the quality of the resulting software. For this reason, the MDD and SOA development approaches should not only be taken into consideration during design and implementation as usually occurs, but also during the RE process.

The contribution of this dissertation aims at improving the development process of health-care applications by proposing OpenUP/MDRE methodology. The main goal of this methodology is to enrich the development process of SOA-based health-care systems by focusing on the requirements engineering processes in the model-driven context. I believe that the integration of those two highly important areas of software engineering, gathered in one consistent process, will provide practitioners with many benefits. It is noteworthy that the approach presented here was designed for SOA-based health-care applications, however, it also provides means to adapt it to other architectural paradigms or domains. The OpenUP/MDRE approach is an extension of the lightweight OpenUP methodology for iterative, architecture-oriented and model-driven software development.

The motivation for this research comes from the experience I gained as a computer science professional working on the health-care systems. This thesis also presents a comprehensive study about: i) the requirements engineering methods and techniques that are being used in the context of the model-driven development, ii) known generic but flexible and extensible methodologies, as well as approaches for service-oriented systems development, iii) requirements engineering techniques used in the health-care industry.

Finally, OpenUP/MDRE was applied to a concrete industrial health-care project in order to show the feasibility and accuracy of this methodological approach.

Resumen

Los dominios y problemas en los cuales es necesario introducir un sistema de información son actualmente muy complejos, siendo su proceso de desarrollo de software, a su vez, de la misma complejidad. Uno de estos dominios es la atención sanitaria (Health-care). Model-Driven Development (MDD) y Service-Oriented Architecture (SOA) son enfoques de desarrollo de software que han surgido de cara a tratar dicha complejidad, reducir tiempo y coste de desarrollo, y aumentar la flexibilidad y la interoperabilidad. Sin embargo, muchas técnicas y enfoques que se han introducido son de poca utilidad cuando las mismas no están previstas de un marco metodológico formalizado y bien documentado. Una metodología provee una estructura bien definida al proceso de desarrollo software facilitando de esta forma un análisis y diseno rápido y eficiente, una implementación sin errores, obteniendo así, un producto software de mejor calidad.

Mientras MDD y SOA están siendo impulsadas de cara a su adopción en la industria del software, hay una cuestión crítica la cual aún no se ha abordado antes de que sus ventajas se hagan plenamente efectivas. No cabe duda de que la ingeniería de requisitos (RE) se ha convertido en una tarea crítica dentro del proceso de desarrollo de software. Los errores cometidos durante este proceso pueden tener efectos negativos sobre las posteriores fases del desarrollo, y en la calidad del software resultante. Por esta razón, los enfoques basados en MDD y SOA no sólo deben tenerse en cuenta durante el diseno e implementación, como normalmente se consideran, sino también durante el proceso de ingeniería de requisitos.

La contribución de esta tesina tiene como objetivo mejorar el proceso de desarrollo de aplicaciones health-care, proponiendo la metodología OpenUP/MDRE. El objetivo principal de esta metodología es enriquecer el proceso de desarrollo de los sistemas health-care basados en SOA, centrándose en los procesos de ingeniería de requisitos en el contexto del desarrollo dirigido por modelos. La integración de estas dos áreas tan importantes de la ingeniería de software, unificadas en un proceso consistente, proporcionará ciertas ventajas a los profesionales de la industria. Cabe senalar que el enfoque que aquí se presenta ha sido disenado para aplicaciones health-care basadas en SOA, sin embargo, también proporciona los medios necesarios para adaptarse a otros paradigmas arquitectónicos u otros dominios. El enfoque OpenUP/MDRE es una extensión de la metodología ágil OpenUP para el desarrollo de software iterativo, orientado a la arquitectura y dirigido por modelos.

La motivación de esta investigación proviene de la experiencia adquirida como ingeniero informático profesional en sistemas health-care. Esta tesina también presenta un estudio completo acerca de: i) los métodos de ingeniería de requerimientos y las técnicas que se utilizan en el contexto del desarrollo dirigido por modelos, ii) metodologías conocidas y genéricas de carácter flexible y extensible, así como los enfoques para el desarrollo de sistemas orientados a servicios, iii) las técnicas de ingeniería de requisitos utilizadas en la industria health-care.
Por último, OpenUP/MDRE se aplicó a un proyecto industrial concreto de health-care a fin de demostrar la viabilidad y precisión de esta metodología.

Resum

Els dominis i problemes en els quals és necessari introduir un sistema d'informació són actualment molt complexos, i el seu procés de desenvolupament de programari, es considera de la mateixa complexitat. Un d'aquests dominis és l'atenció sanitaria (Health-care). Model-Driven Development (MDD) i Service-Oriented Architecture (SOA) són enfocaments de desenvolupament de programari que han sorgit per a tractar aquesta complexitat, reduir temps i cost de desenvolupament, i augmentar la flexibilitat i la interoperabilitat. No obstant aixo, moltes tecniques i enfocaments que s'han introduit són de poca utilitat quan les mateixes no estan previstes d'un marc metodologic formalitzat i ben documentat. Una metodologia proveeix una estructura ben definida al procés de desenvolupament programari facilitant d'aquesta manera una analisi i disseny rapid i eficient, una implementació sense errors, obtenint així, un programari de millor qualitat.

Mentre MDD i SOA estan sent impulsades amb vista a la seva adopció en la indústria del programari, hi ha una qüestió crítica la qual encara no s'ha abordat abans que els seus avantatges sigan plenament efectives. No hi ha dubte que l'enginyeria de requeriments (RE) s'ha convertit en una tasca crítica dins del procés de desenvolupament de programari. Els errors comesos durant aquest procés poden tenir efectes negatius sobre les posteriors fases del desenvolupament, i en la qualitat del programari resultant. Per aquesta raó, la perspectiva basada en MDD i SOA no només s'han de tenir en compte durant el disseny i implementació, com normalment es consideren, sinó també durant el procés d'enginyeria de requisits.

La contribució d'aquesta tesina té com a objectiu millorar el procés de desenvolupament d'aplicacions health-care, proposant la metodologia OpenUP/MDRE. L'objectiu principal d'aquesta metodologia és enriqueixer el procés de desenvolupament dels sistemes health-care basats en SOA, centrant-se en els processos d'enginyeria de requisits en el context del desenvolupament dirigit per models. La integració d'aquestes dues arees tan importants de l'enginyeria de programari, unificades en un procés consistent, proporcionara certs avantatges als professionals de la indústria. Cal assenyalar que l'enfocament que aquí es presenta ha estat dissenyat per a aplicacions health-care basades en SOA, pero, també proporciona els mitjans necessaris per adaptar-se a altres paradigmes arquitectonics o altres dominis. L'enfocament OpenUP / MDRE és una extensió de la metodologia agil OpenUP per al desenvolupament de programari iteratiu, orientat a l'arquitectura i dirigit per models.

La motivació d'aquesta investigació prové de l'experiencia adquirida com a enginyer informatic professional en sistemes health-care. Aquesta tesina també presenta un estudi complet sobre: i) els metodes d'enginyeria de requeriments i les tecniques que s'utilitzen en el context del desenvolupament dirigit per models, ii) metodologies conegudes i generiques de caracter flexible i extensible, així com els enfocaments per el desenvolupament de sistemes orientats a serveis, iii) les tecniques d'enginyeria de requisits utilitzades en la indústria health-care.
Finalment, OpenUP/MDRE es va aplicar a un projecte industrial concret de health-care per tal de demostrar la viabilitat i precisió d'aquesta metodologia.

Acknowledgements

First and foremost, I would like to express my heartfelt gratitude to my supervisor prof. Emilio Insfran for his encouragement, patience, expert advices and without whose invaluable help this master thesis could not have been completed.

I also acknowledge my gratitude to prof. Silvia Abrahão and Ausiás Armesto for their valuable assistance, comments and suggestions.

I would like to extend my sincere thanks to my bosses and friends from Indra Sistemas: Damian Vidal Segui and Juanjo Cubillos Esteve who helped me in my efforts with their comprehension and advices. Without them, I would not have gained so many valuable experiences.

At this moment, I am particularly grateful to my mom and dad, who from the distant Poland always have supported me with their thoughts and prayers.

Special thanks to all members of the ISSI research group for their helpful hand which they never refused. I thank them also for creating a friendly atmosphere that I have been noticing during my master studies.

I would also like to thank all my friends from Valencia, in particular my best friends Rodrigo and Primi, who supported me everytime their support was needed.

I am very grateful to them all.

Chapter 1

Introduction

Software systems are becoming more and more complex, and the success of their development no longer depends on individual effort and heroics. Successful software development can only be accomplished by using a well-defined software development process. Many software development approaches have been introduced to deal with problems complexity, reducing the time and the cost of development. However, these techniques are of little use if well-defined guidelines are not provided to assist software engineers in the software development process. The goal of the present master thesis is to define a methodology which incorporates the requirements engineering techniques to a well-specified model-driven process, which should facilitate the process of developing a complex software.

This chapter is organized as follows. Section 1.1 presentes the motivation of the research. Section 1.2 identifies the principal goals of this master thesis, pointing which are the associated tasks to reach the objectives and the expected result. Section 1.3 places this research in a particular context of its development. Finally, Section 1.4 describes the structure of the present thesis.

1.1. Motivation

The great motivation of this research is the application of the Model-Driven Development principles at all possible software development process stages. Recent investigations of the industry and also academia have as their objective to provide: methods, tools, and standards, that could improve the software development process in the terms of its quality, simplicity of use, flexibility of adapting to different technologies, as also speeding up the time-to-market of its products. For this reason the Object Management Group (OMG) came up with the concept of Model-Driven Architecture (MDA). How-

ever, the focus of MDA is on standardization of notations and on tool interoperability. The OMG offers little in terms of methodological support for model-driven software development. Thus, tool vendors define their own approaches, typically addressed to some specific tools rather than providing comprehensive support for an end-to-end software development process.

It is beyond dispute that a well-defined software development process is essential for the final project success of each project and the requirements engineering is the critical task within this process. Errors made during this process may have negative effects on subsequent development steps, and on the quality of the resulting software.

However, software development engineers creating software with which to support many complext domains, such as health-care, do not pay enough attention to the requirements engineering tasks. These tasks demand a carefull and cautious management within the development process. Health-care is domain where the supporting software has to be perfectly designed in order to provide systems interoperability, integrate various devices and distributed software systems, execute complex automated or interactive business processes, while fully satisfying all the stakeholders requirements and keeping the software understandable and user-friendly. Health-care is today one of the fastest growing economic sectors, where on the other hand huge amounts of money are wasted because of the errors at the requirements engineering stage. This is where the Software Process Engineering (SPE) has an important role to play.

The methodological approach which is the subject of this thesis is a solution that facilitates the work of software engineers developing health-care middleware platforms, as my main experience comes from this field. Several years of experience in the domain allow me to identify the poor points of the health-care software engineering and propose an approach with which to improve the development process in the domain.
The methodological approach presented in this thesis defines innovative software development process in which the requirements engineering is placed in the model-driven context. I believe that the integration of those two highly important areas of software engineering, gathered in one consistent process, will provide practitioners with many benefits.

1.2. Thesis objectives

The principal objective of this master thesis is to propose a requirements engineering model-driven approach for the development of complex systems in the health-care domain. To reach this goal, the work has been decomposed into tasks to be performed within this research. These tasks are the following:

1. to perform a systematic literature review (SLR) with which to analyze current use of the model-driven approaches which cover the requirements engineering activities; this SLR has to give an answer to the following research questions: "what

requirements engineering techniques have been employed in model-driven development approaches and what is their actual level of automation";
2. to investigate on the current approaches of requirements engineering field in the health-care domain;
3. to define a generic methodology that builds upon the model-driven development principles, and facilitate the requirements engineering discipline;
4. to adapt the proposed methodology for the purposes of the SOA-based health-care middleware;
5. to implement the methodology in a process engineering tool with which to provide the process definition easily accessible by developers;
6. to develop an industrial case study with which to demonstrate the feasibility of the approach;

1.3. Research context

The present master thesis has been developed based on the experience gained as a computer science professional.

- I+D developer at **Dimension Informatica** (Valencia, Spain) with the main responsibility of improving the software development methodology of the organization as a member of the Methodologies and Technological Innovation Department.
- senior developer position at **Indra Sistemas** (Valencia, Spain) with the main responsibility of providing a methodological support for I+D research projects, such as: MPOWER, Tratamiento 2.0, with the focus on the requirements engineering and architectures development.

This investigation project started in 2008 being a part of a larger research on the health-care systems development methodologies. It constitutes a contribution to the following investigation projects:

- **MPOWER** Project: Middleware platform for empowering cognitive disabled and elderly (October 2006 - September 2009). Project (with ref. 034707) funded by the European Union under 6th FWP (Sixth Framework Programme).
- **TRATAMIENTO 2.0** Project: Generic middleware platform for tele-management of intelligent medical treatment (January 2008 - December 2010). Project funded by the European Commission (European Regional Development Fund), "The Ministry of Industry, Tourism and Trade (Spain)", "Plan Avanza2", "Plan Nacional de I+D+i 2008 - 2011"
- Collaboration as invited researcher in the Multi-modeling Approach for Quality-Aware Software Product Lines (**MULTIPLE**) Project with ref. TIN2009-13838 (October 2009 - September 2013) funded by the Ministry of Science and Innovation (Spain)

1.4. Structure of the document

Chapter 2 gives an overview of the background concepts of this work. It includes a short description of the basic software engineering approaches which are gathered into one joined approach in created new methodology. Moreover, it describes two methodologies (RUP, OpenUP), which form the basis for the present work. Also the health-care domain is introduced as it is a target domain for the methodology to apply. Chapter 3 presents an investigation with which to conclude the existing works in the field of model-driven development, requirements engineering, service-oriented approaches, and health-care domain information systems. In particular a systematic literature review is presented on the use of the requirements engineering techniques within the model-driven processes. Chapter 4 describes the methods employed in this work in order to meet the established research goals. Chapter 5 constitutes the main part of this work describing the new methodology OpenUP/MDRE. It includes the description of all important elements of a methodology definition, such as: artifacts, roles, tasks, activities, and their workflows. Chapter 6 contains a case study with which to show the accuracy and simplicity of the methodology use. The case study shows the OpenUP/MDRE application in the health-care domain by its flexible adaptation to the specific project architecture (SOA). The example provided is based on a real industry project from the health-care domain. Finally, the last chapter concludes this master thesis prviding some final remarks on what has been achieved and identifying directions of some further works.

In addition, Appendix A provides definitions of all concepts which are necessary to understand the subject of this work. It includes terms, mostly related to elements of the methodology definition, but also concepts commonly used to describe model-based or model-driven software engineering approaches.

Chapter 2

Research Background

In the recent decade, computer science investigation bodies introduced many software development approaches intending to help developers in producing software. One of such approaches is Model-Driven Engineering (MDE). MDE is a software development methodology which focuses on creating models, or abstractions, closer to some particular domain concepts rather than computing concepts. Model-Driven Architecture (MDA) is a standard realization of the MDE by the Object Management Group (OMG) which specifies three modeling abstraction levels for the development lifecycle. In this context, the Model-Driven Development (MDD), also called Model-Driven Software Development (MDSD), is a new software development paradigm for creating working software taking the advantage of models and model transformations. The MDD priority is to provide scalability and also to facilitate the software validation by end users and stakeholders as early as possible. According to the MDA framework and the MDD approach, a solid and complete development process was defined.

The proposed methodological solution was created on the basis of a real industrial project of a health-care middleware platform development. This project was based on another important approach called Service-Oriented Architecture (SOA), which provides a flexible set of service-based design principles for distributed, interoperable, secure, etc., applications. SOA is currently very frequently used especially in the health-care. With the purpose of joining the best software development approaches, such as: MDD, RE techniques, and SOA, a new methodology, that finds its application in highly complex domains (such as health-care), was created. To date, the health-care is still lacking in a complete methodology which would effectively support the software development.

However, defining a methodology from the scratch is very laborious and difficult task. That is why many methodologies are built with reusable elements and structured

building blocks, additionaly providing extensions mechanisms. Two methodologies with aforementioned characteristics are used in this work to construct the final research result. The Rational Unified Process (RUP) which possess well-defined process to treat requirements during the development process and the OpenUP methodology which emphasizes the importace of an architecture in the development process.

This chapter is organized as follows. Section 2.1 introduces the requirements engineering (RE) as a essential part of the software development process. Section 2.2 introduces the MDE, in particular the Model-Driven Architecture, its principles, abstraction levels that the architecture defines with its framework structure. Section 2.3 characterizes the health-care domain, its uniqueness and complexity, to justify why this particular domain was chosen for this new requirements engineering approach in the model-driven context. The following section 2.4 presents some basic principles of the Service-Oriented Architecture (SOA), which currently is the most popular approach for creating health-care systems. Finally, section 2.5 gives an overview of the Rational Unified Process with its main building blocks, such as: iterations, phases, and disciplines. Moreover, a more detailed description of the requirements discipline, which includes: roles, work products, activities, and workflows, is provided as its workflow constitutes the base of the new model-driven requirements approach. Section 2.6 describes the OpenUP methodology. It being built as a part of the Eclipse Process Framework Project, perfectly matches with the goals of the new agile methodology. Finally, the last section introduces the methods engineering principles by discussing known standards and tools that help in effective generic methodologies adaptation.

2.1. Requirements Engineering

Requirements Engineering (RE) is a specific discipline of the software engineering. The RE process is recognized as being the most critical process in software development. Errors made during this process may have negative effects on subsequent development steps, and on the quality of the resulting software. In the RE we can distinguish two groups of activities as shown on the Figure 2.1. The first group is related to the requirements development and the second group gathers activities classified as requirements management activities. The former group contains activities for requirements elicitation, analysis, specification, and validation. On the other hand, requirements management covers establishing and maintaining an agreement with stakeholders on the requirements, controling the baselined requirements and their changes, and finally keeping requirements consistent with plans and work products.

Westfall in [75] describes the requirements engineering principles answering the following 5 questions:

- **What?**

  The answer to this question describes the various types of requirements to be

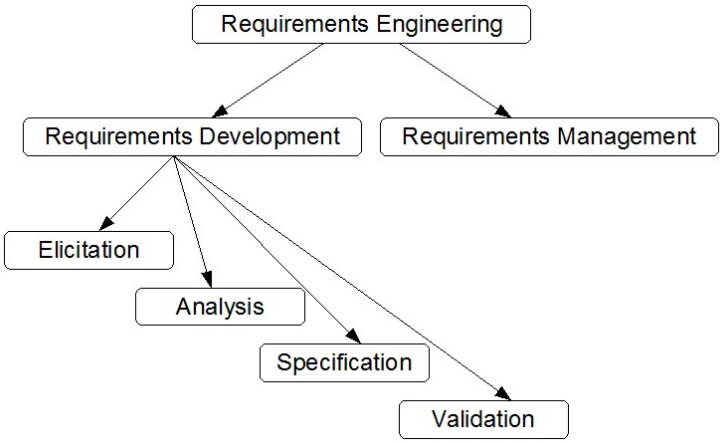

Figure 2.1. Requirements Engineering subdisciplines

defined. *What* the software must be, *what* it should be like and *what* kind of limitations there are regarding its furthere implementation. Giving an answer to these questions you describe functional and non-functional requirements as also the external interface definitions and constraints. Figure 2.2 presents a taxonomy of the possible types of the requirements specification.

- **Why?**
  The *"Why"* question emphasizes the importance of the requirements engineering tasks with regard to the entire software development process. The simple answer is that requirements engineering provides means to describe correctly the stakeholders needs. Even a perfect development process will not help if you are developing the wrong product.
- **Who?**
  The requirements engineering gives the opportunity to involve the stakeholders in the development process. The lack of user involvment is a frequent reason of developing a wrong product.
- **When?**
  Requirements activities should be performed throughout the entire software development lifecycle. RE is an iterative process. It starts with the elicitation of a set of consistent and complete requirements. But of equal importance is their refining and managing changes to these requirements once the development process has been launched.
- **How?**
  In last decades, many approaches have been introduced for eliciting, analyzing, specifying and validating software requirements. One of such approaches is the Model-Based Requirements Engineering (MBRE) which promotes the use of models as a primary artifacts of the requirements engineering.

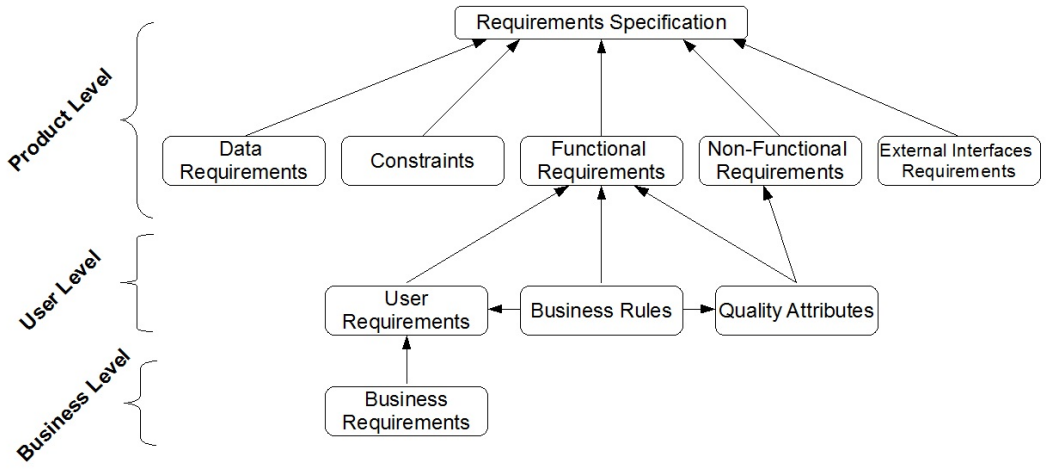

Figure 2.2. A taxonomy of the requirements specification types

2.2. Model-Driven Engineering

Model-Driven Engineering (MDE) is a software development methodology which focuses on creating models as a primary form of expression and primary artifacts of software specifications. This kind of specification is closer to the particular domain concepts than the specifications focused on algorithmic solutions. The abstraction of concepts leads to increased productivity by maximizing compatibility between systems, simplifying the process of design, and promoting communication between individuals and teams working on the system.
Models can be used a means to better understand the problem domain, but can also be a part of a more complex process of automated code generation. Depending on the level of detail, the code can be generated from the models, ranging from system skeletons to complete, deployable products.
MDE technologies with a greater focus on architecture and corresponding automation bring higher levels of abstraction in software development. This abstraction promotes simpler models with a greater focus on problem space. Combined with executable semantics this elevates the total level of automation possible.

In this context, the Object Management Group (OMG) has developed an advanced architecture-focused approach calledf the Model-Driven Architecture (MDA), which to date is the best know MDE approach. This kind of initiatives that promote models, modeling, and model transformations form a set of software development approaches known as Model-Driven Development (MDD) approaches.

2.2.1. Model-Driven Architecture

The *Model-Driven Architecture* (MDA) is an approach to IT systems specification defined by the *Object Management Group* (OMG). MDA consists of a set of guidelines

which support *Model-Driven Engineering* (MDE). These guidelines mainly concern the structure of software specifications in the terms of models and their transformations. OMG has also defined a standard language for model transformation called *Query View Transformation* (QVT).

MDA separates the system functionality specification and its further implementation on a specific platform. MDA distinguishes three abstraction levels of models, which are described in the following sections. This specification is based on the concept of a model that is considered to be the basic and most important specification structure unit.

Through such separation of the specification from the implementation technology platform, the specification can be reused for different technologies and multiple platforms. This provides a flexibility on changes in the implementation technologies or adopting the system functionality to the current standards. Different applications can be integrated with prepared models through applying mappings to specific platforms. Figure 2.3 shows how the MDA approach fits in the general image of software development lifecycle and its adaptation to any kind of technologies in different domains. The figure shows how the process spreads from the core implementation of models to the specific implementations using different technologies, programming languages and platforms.

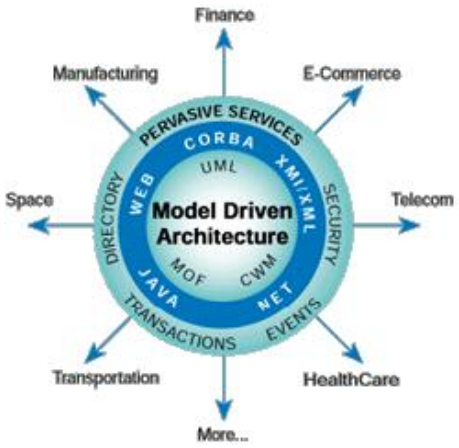

Figure 2.3. MDA lifecycle

2.2.1.1. Computation-Independent Model

Computation-Independent Model (CIM) is a business model specification which usually describes a system from the organizational point of view. It specifies business processes, organizational structure of the company or stakeholders participating in the processes. Information about the software systems is usually not included that is why the computational independence is emphasized. The CIM if specified is normally the starting model of the projects specification. The requirements for a software systems are derived from CIM based on the business models it provides. Each business model

from the CIM has associated its specific technical, software model where the software model is a description of the software system from the functional point of view.
CIM represents business context for the IT Solutions, expressed in terms of business processes and business concepts. It constitues the basis for creating more specific analysis models, closer to the technical development process. However, performing automatic CIM transforations is often a very difficult or even impossible, becuase of the lack of formality of its representation. CIM-level model transformations usually demand a human to be involved in this manual or semi-automatic transformation process.

2.2.1.2. Platform-Independent Model

Platform-Independent Model (PIM) describes business functionality and behavior in automated way, but undistorted by technology details. PIM uses an appropriate domain specific language for system functionality definition.
The specification of PIM usually consists of diagrams of different types (frequently used is the Unified Modeling Language (UML)), to model the functional requirements in a technology independent manner. PIM is a description of a system that realizes some business defined at the CIM level. The technological solutions are not important at this stage of the MDA-based modeling. PIM is the most important level of models in the model-driven architecture as it provides models to be later transformed for a particular technology.
To PIM can be applied various transformations which create new models either at the same abstraction level or passing to the next PSM level. The former transformations from PIM to PIM are called endogenous transformations, meanwhile the latter are called exogenous. PIM to PIM transformations allow to generate different models considering the same functionalities specification from different point of views and also focusing on different aspects of that specification. Depending on the transformations to be applied to the PIM models, the required/recommended model types can be different.

2.2.1.3. Platform-Specific Model

Platform-Specific Model (PSM) is the functional specification of a system according to the specific implementation technology and its constructs and limitations. PSM is produced from PIM models after applying a number of different types of transformations. A PIM can be transformed into one or more PSMs. For each specific technology platform a separate PSM is generated. PSM facilitates the work of developers in the implementation phase when PSMs offer the possibility of automatic code generation out of the specification considering platform and technology design details.

2.2.1.4. MDA development lifecycle

Aforementioned modeling abstraction levels introduced by MDA standard can be applied on different software development stages to describe different development areas,

such as: user requirements specification, business processes, or functional detailed specification.

The MDA proposes the following lifecycle of aforementioned models. The software development starts with the CIM which also sometimes is considered as not necessary and skipped by the developers. CIM usually lacks in formality and is very often represented by natural language specifications or different kind of graphs, what makes it more difficult to introduce an automated process of the CIM transformations. Next the PIM for an application's business functionality and behavior is constructed using a modeling language based on OMG's *Meta Object Facility* (MOF) [62]. The idea is to use stable models meanwhile technologies may evolve and change. Later the PIM is converted to the PSM. It means that a new model at the PSM-level is constructed which will be used with a specific technology platform. The key point of the MDA framework is that for one PIM, many PSMs, for different user environments and technological platforms, can be created. One functional model can be transformed to match with different platforms and generate adequate implementation, such as: Web Services, XML/SOAP, EJB, .Net, CORBA, etc., for a corresponding platform. The Figure 2.4 shows the relation between abstraction levels described in the previous sections and graphically demonstrates discussed here MDA framework lifecycle.

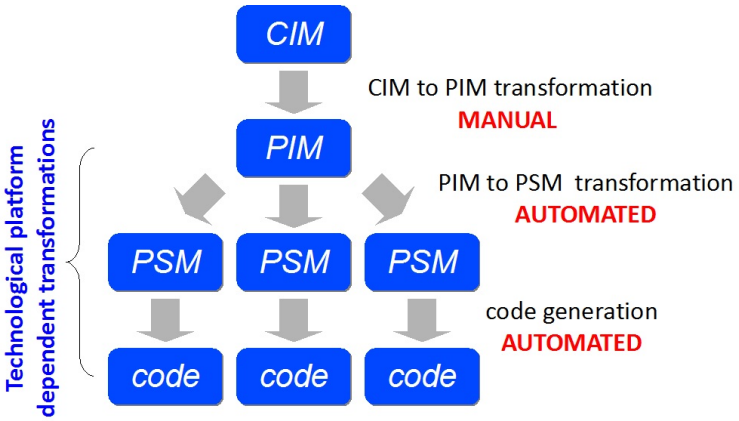

Figure 2.4. MDA framework schema

The interoperability in the MDA is guaranteed by the usage of industry standards for modeling specifications. These standards are: MOF, UML v2.0., the Common Warehouse Metamodel (CWM), XML Metadata Interchange (XMI). MDA assumes that the PIM representing the conceptual design through different models will stay the same independently on realization technologies and architectures. Such decoupling of these two domains provides developers with the possibility of changing the realization technology as the world of technology gets more reach with time. Also some particular models can be chosen for a specific domains that fits better the modeling one.

Currently there are many tools called "MDA tools" that support described here lifecycle by providing means to develop, interpret, compare, align, measure, verify, transform, etc. models or meta-models. MDA tools work with two types of models: initial models created by analysts in a modeling language and derived models generated by applying different transformations. The latter can be helpful in further code generation.

2.2.2. Model-Driven Development

Model-Driven Development (MDD) is a paradigm for writing and implementing computer programs quickly, effectively and at minimum cost. It describes a family of approaches that use models and model transformations to create software products. MDD is the next step of abstraction in writing software applications. However, MDD does not always prove popular with developers which prefer to code rather than build models. The modeling activity can be more frequently found in large scale projects of large enterprises of highly complex domains, such as: defence, aerospace, or health-care.

In the MDD code is automatically or semiautomatically generated from more abstract models, and which employs standard specification languages for describing those models and the transformations between them. It also supports model-to-model transformations. Brown et al. in [11] argue that models and the transformations that relate different models are of equal importance for the productivity of the software development. The MDD process and its difference between the traditional development process shows the Figure 2.5. This MDD is built on the basis of the MDA principles making use of models and model transformations.

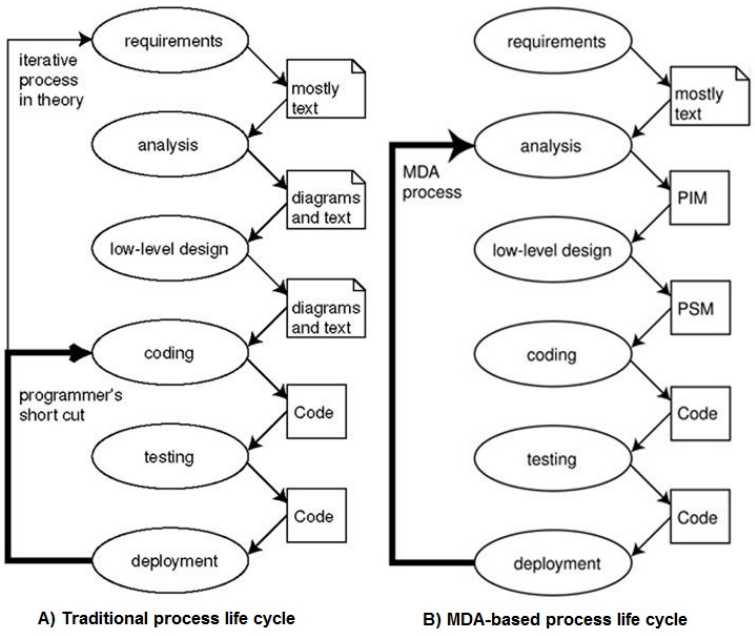

Figure 2.5. Traditional and MDA-based processes comparison

2.3. Health-care

Highly complex processes, huge number of multi-stakeholder dynamic requirements, great role of non-funcional requirements, such as: usability, interoperability, security, political and legal issues, these are only few characteristics describing the health care domain. Software quality is especially important in this domain where it is possible that badly designed software systems can significantly harm patients, causing even patient's death. Producing quality software demands also the development process to be of high quality. That is why this domain was chosen to evaluate the methodological approach presented in this work.

This section presents the health care domain, its main characteristics and uniqueness in the context of the complexity of its requirements in software development process. Requirements in health care are critical because produced software involves peoples life and thus is every sensitive to every error and mistake commited in development process, especially in the requirements phase.
As comment Garde and Knaup in [28] errorless software is a must in health care where requirements engineering is an extremely time-consuming and complex process which needs specialized methods and approaches. They also identify some main characteristics of the domain.

- **The domain complexity (clinical knowledge, data to be stored, care conditions, care relationships) is very high.**
  Building health care systems require a great specialized knowledge of the domain. At the same time this knowledge is very dynamic and constantly changes. Data on which operate health care systems are complext and of different kinds. Also the conditions of care are complex what means that many care decisions depend on many different conditions for different users and contexts.
- **The stakeholders are very complex.**
  In the patient's care many different stakeholders are involved, each of which has different requirements and priorities. The majority of them are non-technical professionals. There are many challenges in performing RE activities for multiple stakeholders, such as: different perspectives on the health care system, variety of different: objectives, goals, needs, expectations, perspectives, priority of requirements of each stakeholder, stakeholders can be of different backgrounds which can cause communication problems. In addition, not only care providers are the stakeholders involved in the care process, but also people from the patient environment and social network, they maight be also considered as the health care system users. That is why it is very important to properly elicit requirements which would not be possible without the stakeholders participation in the health care requirements elicitation process. There are approaches focused on including stakeholders in the design of e.g. home care systems such as the approach presented by McGee-Lennon and Gray in [54].

- **The variability of health care requirements, contexts and relations between them.**
  Each patient has an unlimitted set of characteristics, moreover these characteristics constantly change. Health care systems have to deal with situations which are sometimes not well known and predicted. Many characteristics are strongly related to other characteristics creating a unique for the situation context. These interactions beetween requirements and contexts influence the domain complexity and make that the variability factor is very high.
- **The relevance of health information over time is very important.**
  The health care systems should be able to cope with dynamically changing requirements of different stakeholders. Patient-centric character of such systems requires adopting the system performance to changing contexts. The health care information that changes over time should be systematically collected and updated.
- **Requirements come from different distributed sources.**
  In health care the information may come from a variety of sources. The conjunction of that information can be used to describe complext correlated requirements. For example in home care information has to be taken from the patient's home environment and health care centers building together the picture of the home care process.
- **Patient safety is the crutial factor to have in mind.**
  Health care is one of these domains where errors, misunderstandings, system failures are not allowed as we treat with patients lifes. Al medical systems are required to be error-free and medical devices to run flawlessly. In health care many conditions have to be fulfulled simultaneously and each of these conditions is very complex. That is why the health care systems development process should be focues to produce harmless software taking into account all physical and psycological aspects.

In health care many non-functional requirements (NFR) play an essential role in the patient care process, they are not only important but also numerous. Mairiza et al. in [49] presented a systematic literature review on NFRs and their use. One of research questions used for the review was: "Which types of NFRs are of concern in various application domains" giving as a result, among other domains, crutial NFRs for medical/health care domain. These NFRs are the following: communicativeness, confidentiality, integrity, performance, privacy, reliability, safety, security, traceability, and usability. There are works emphasizing the role of non-funcional requirements in the health care like Cysneiros in [15] focusing especially on safety, security, privacy, reliability, availability and usability.

Within the RE process, unknown or not completely understood requirements are the main problem. That is why there are many approaches to the elicitation of require-

ments in the health care. For example Cysneiros in [15] discusses various elicitation techniques in health care. He also points that requirements are very often conflicting among them and these conflicts should be carefully searched in the elicitation phase. One specific part of the health care domain is home care. Many current researches in the health care field are related to home care, as large number of people (e.g. elderly or cognitively disabled) prefer to stay at home to receive care. This is socially beneficial, as they can stay at their familiar environments and also economically beneficial. The challenge in this case is to provide complete and professional care outside the health centers. McGee-Lennon in [53] suggests features which should be provided by requirements engineering approaches in the home care requirements elicitation and analysis. However, these good practicies can be also applied in general to the health care domain RE approaches. Regarding these suggestions the elicitation of health care requirements should be an interative process, balancing and validating requirements. This process should also include active participation of interested identified stakeholders to elicit high quality requirements. Next, prioritisation of requirements as well as identification and categorisation are considered to be good practices. Requirements traceability should be also provided as well as resolution of requirements conflicts.

The complexity of the domain and its unique character nicely summarizes Jones in his paper: "Computers can land people on Mars, why can't they get them to work in a hospital?" [35]. That is why McGee-Lennon [53] argues that novel, health care dedicated RE approaches are required in order to improve the quality of patient's care.

2.4. Service-Oriented Architecture

Service-Oriented Architecture (SOA) is an architectural style that emphasizes loosely coupled, course-grained, sharable, secure, network based services to enable flexibility in an interoperable technology agnostic manner [22]. As a flexible and extensible architectural framework, SOA has the following unique capabilities:

- **Reduced cost** - the cost of developing SOA-based applications is minimized as only necessary functionalities demanded by users are implemented; functionality decoupling into services and services management are the means to avoid maintaining redundant and obsolete functionalities;
- **Increased flexibility** - applications based on loosely coupled services facilitate the rapid restructuring and reconfiguration of the business processes and applications that consume these services.
- **Increased interoperability** - web services (WS) are the most frequently used services implementation; the use of standard protocols makes these implementations interoperable with many existing business applications and thus provides the opportunity to enter into new markets offering new business services;

- **Added agility** - speeds up the time-to-market of business applications as they are constructed as orchestrations of loosely coupled services;
- **Increased consolidation** - integrates systems across many geographically distributed organizations;

The SOA framework is build of many important concepts, which have been gathered into a reference architecture by the IBM Service-Oriented Architecture experts. They defined an SOA reference architecture based on their experience from multiple projects in various industries.
The reference architecture defines the layers, architectural building blocks, architectural and design decisions, patterns, options, and the separation of concerns that are helpful in implementing SOA. The Figure 2.6 shows a simplified version of the IBM SOA reference architecture. It consists of layers, which facilitate separation of concerns and assist the process of creating an SOA applying different kind of service-oriented techniques, such as the Service-Oriented Modeling and Architecture (SOMA) method.

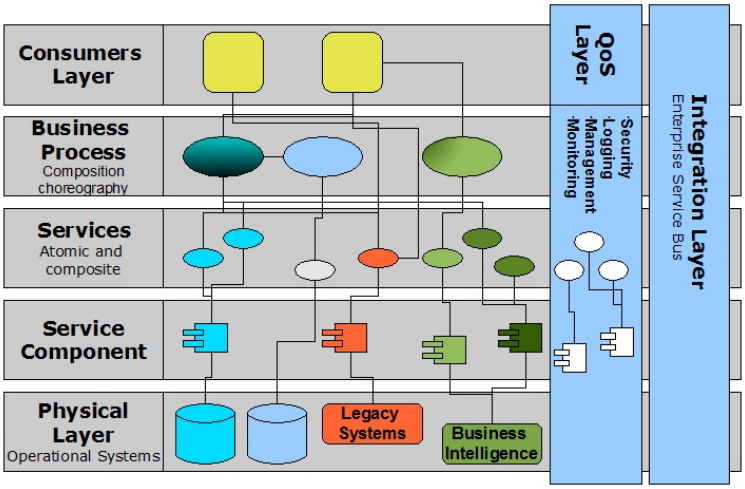

Figure 2.6. IBM SOA reference architecture

The SOA reference architecture consists of the following layers:

- **Physical/Operational Layer** - This layer is made up of existing application software systems. A number of existing software systems are part of this layer. Those systems include: monolithic custom built applications (J2EE™ and Microsoft® .NET®), legacy applications and systems, transaction processing systems, databases, and packaged applications and solutions (including ERPs and CRM packages, such as: SAP or Oracle solutions), as well as business intelligence applications. The composite layered architecture of an SOA can leverage existing systems and integrate them using service-oriented integration techniques.
- **Service component Layer** - This layer contains software components, each of which provide the implementation for, realization of, or operation on a service.

Service components reflect the definition of a service, both in its functionality and its quality of service. This layer typically uses container-based technologies such as application servers to implement the components, workload management, and its high-availability.

- **Services Layer** - This layer consists of all the services defined within the SOA. A service is considered to be an abstract specification of a set of functions that satisfy some business requirements. The specification provides consumers with sufficient detail to invoke the business functions exposed by a provider of the services, which can be discovered or statically bound, to be later invoked or choreographed into a composite service. The information about the service functionality can be specified in a Web Services Definition Language (WSDL).
- **Business process Layer** - also called composition or choreography layer. Compositions and choreographies of services exposed in the *Services Layer* are defined here. Groups of services can be combined into flows establishing the performance of an application. The entire use cases and business processes can be implemented by orchestrating services into flows. To do this, visual flow composition tools can be used for design of application flows.
- **Consumers/presentation Layer** - This layer consists of SOA-based applications which make use of all previously commented layers. It provides the capability to quickly create the front end of business processes and composite applications from loosely coupled services and their orchestrations.
- **Integration Layer** - This layer enables the integration of services through the introduction of a reliable set of capabilities, such as intelligent routing, protocol mediation, and other transformation mechanisms, often described as the Enterprise Service Bus (ESB). WSDL specifies a binding, which implies a location where the service is provided. On the other hand, an ESB provides a location independent mechanism for integration.
- **Quality of service Layer** - This layer provides the capabilities required to monitor, manage, and maintain QoS, such as: security, performance, and availability. This is a background process through sense-and-respond mechanisms and tools that monitor the health of SOA applications.

2.5. Rational Unified Process

Rational Unified Process (RUP) is a methodology focused on creating a high quality software projects which have to be done in some predefined period of time, by the means of certain amount of money and have to be compatible with the specified user requirements. RUP exactly defines who is responsible for what, when and how different activities should be done. It also provides well specified structure of the project development lifecycle. This methodology suggests to follow some practices of project's

documentation. As a result a huge amount of documentation is created, but such well documented process helps in projects management and leading of an unexperienced team. The basic characteristics that describe the Rational Unified Process are listed below.

- It is architecture oriented. Architecture is the basic element of the process based on RUP. This process is analyzed, constructed and managed. Planning and team management is a frequent activity as the constructing system is divided into subsystems and layers, and all those parts demand their separate control. The architecture also points which elements of the developing system are reusable or which are third party elements.
- RUP defines an iterative development process what means it is divided into series of iterations. During each iteration, activities belonging to many disciplines are executed. The scope of those activities depends on the project development phase and the project's current stage. Iterativity has many advantages in front of the classical development process.
  - The final product quality is high because of executing different types of tests at the end of each iteration. Also the iterative process gives the possibility to capture new requirements more precisely and validate implemented functionality against those already existing.
  - Relatively quick problems detection gives the possibility to take some preventing actions in case of those emergency situations.
  - Constant integration helps to avoid time consuming integration process which regarding the classical model is executed at the end of system development. Here the integration process takes place at the end of each iteration integrating newly created components with those previously implemented.
  - Thanks to iterativity the reusable elements can be easily identificated because of taking advantage of already captured requirements and already implemented functionality from the previous iterations.
- RUP is use case driven. Use cases describe the system functionality from the user's point of view. Their description is understandable as well for the team of developers, as for the client side. Use cases are easily traceable in different kind of models, user requirements and artifacts such as system prototype or tests. Use cases establish the base for the development process.

RUP is a complete methodology, that means it defines the complete development process from user requirements elicitation to the product deployment end-user environment. RUP divides the work to activities which constitute the development process workflows. These activities are thematically classified to disciplines. Activities from different disciplines can be performed simultaneously. In addition, to structure the

work, RUP introduces such concepts as an iteration and a phase. Moreover, it defines roles to which the particular activities are assigned. Finally, RUP indicates which are the process input and output artifacts. The following subsections describe in more detail the RUP work breakdown structure, in particular two disciplines which are crutial for this research.

2.5.1. Lifecycle

The RUP methodology defines an iterative model of software development. These iterations are classified into four phases: *Inception*, *Elaboration*, *Construction*, and *Transition*, which constitute the principal stages of the software development. The Figure 2.7 shows these phases and the relation between them in the software development lifecycle. Each phase is supported by the product generated in the previous phase. Phase's result can be either a document, a piece of code of implementation or other artifact important for the software project development process. A short description of each of the phases is presented next.

- **Inception** - this phase focuses on the description of the project scope and understanding of the general project's goal and requirements.
- **Elaboration** - focuses on the requirements, its understanding and use as a technical specification. Architectural solution is designed and implemented in the form of a prototype. This phase also covers tasks of technologies and tools investigation for the project's purpose.
- **Construction** - focuses on the implementation and testing of all the components and features described in the specification. One of the most important management tasks in this phase is the control of resources, costs, schedules and the quality of produced software.
- **Transition** - focuses on applying created software in the real life environment by the end users. This phase considers that the created software moves by parts to the user waiting for the acceptance, and giving feedback to developers about found problems and errors which leads to releasing new software versions.

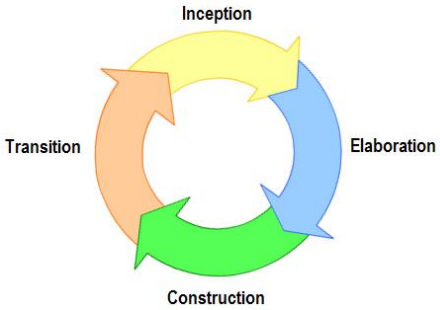

Figure 2.7. Phases of RUP methodology

As mentioned before, RUP organizes its thematically related activities into disciplines. A discipline describes an area of concern within the methodology. RUP defines 9 disciplines which are divided into 2 groups of core and supporting disciplines. The Figure 2.8 presents a hump chart of these two groups of disciplines in the context of the process structure (iterations and phases). In each phase tasks from different disciplines are executed. The hump chart also demonstrates each discipline with the intensity of use of the activities from each discipline during different development phases. It is important to notice, that the workload and the time spent on particular activities is different for each discipline in each of the four phases.

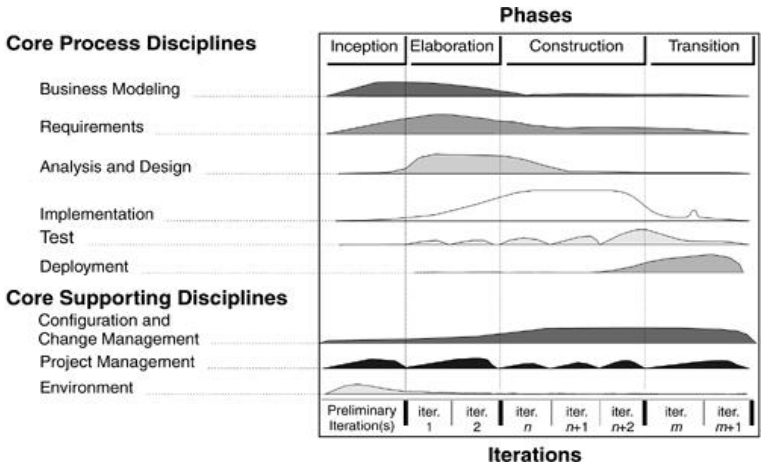

Figure 2.8. Core and supporting RUP disciplines

Two RUP disciplines that will be the subject of this work in further sections are: the *Requirements* and the *Environment* discipline. A short characteristic of the main purposes of these two disciplines is presented below.

Requirements

- to establish and maintain agreement with the customers ajd other stakeholders on what the system should do;
- to provide system developers with a better understanding of the system requirements and stakeholders needs;
- to define the scope of the system;
- to provide a basis for project management tasks, such as: planning the technical contents of iterations, estimating cost and time of the system development;
- to define user interface for the system, focusing on the users needs;
- to manage requirements by eliciting, documenting, and maintaining of requirements through the entire development process;
- to trace requirements by verifying that all requirements of the system are fulfilled, and that the application does only what it was intended to do;

Environment

- to configure the process by its adaptation to the needs of the organization by altering the process defined by RUP;
- to implement the process by changing the organization's practice so it uses the RUP in a part or in whole;
- to select and acquire tools that fit the particular needs of an organization, based primarily on specific activities or artifacts necessary for the process. Aforementioned tools can be tools for modeling, requirements management, code development, testing, etc.;
- to develop tools, sometimes special tools must be developed internally to support special needs providing additional automation of tedious or error-prone tasks;
- to support the development providing different technical services, such as: maintaining the hardware and software development environment, system administration, upgrades, etc.;
- to perform training, depending on the complexity of the organization's software development process, special training may be necessary to educate developers on the process, tools and techniques;

2.5.2. Requirements discipline

The main objective of this discipline is the stakeholders' acceptation on the captured requirements which have to be fulfilled in the software to be created. The description of requirements makes system developers understand better what the system should do and how exactly it should work. The project limitations are defined and agreed. Moreover artifacts created within this discipline provide information for further planning of work schedule, estimation of project costs and the more important thing that provide a basis for analysis and design phase and their technical content. This section describes the requirements discipline with its workflow of activities. Moreover, it discusses related elements, such as: activities, roles, and artifacts, that form the description of the RUP methodology.
Artifacts of this discipline describe functional and also non-functional requirements. The latter include a description of the environment in which the future system will be used. This description is mainly focused on some considerable parameters such as scalability, security, backup and recovery of the system, etc.

2.5.2.1. Roles

There are two roles taking part in the requirements discipline the **System Analyst** and the **Requirements Specifier**. The *System Analyst* leads and coordinates requirements elicitation and use-case modeling by outlining the system's functionality and delimiting the system. The *Requirements Specifier* details all or part of the sys-

tem's functionality by describing the requirements aspect of one or several use cases. The Figure 2.9 shows the relation between these roles and associated artifacts.

2.5.2.2. Artifacts

The *Requirements* discipline defines the following artifacts:

- **Requirements Management Plan** - describes which are the requirements artifacts, their types and attributes. It contains a plan of requirements to be collected and how they will be later managed by the means of control mechanisms for measuring, reporting and controlling changes.
- **Stakeholder Requests** - gives an overview of what each of different stakeholders expect from the system and how each stakeholder wishes the system to be look like.
- **Vision** - one of the most important artifacts not only of the requirements discipline, but rather of the entire software development process. The Vision document describes the main system features and requisites. To describe details of the system functionalities additional documents with specification are provided. The Vision Document is focused on the customers' perspective, discussing the essential features of the system and its acceptance levels. The document clearly defines the scope of features to be implemented. This document should also describe system users and their operational capacities. At the end the Vision Document stands as a base of requirements' documents, understandable and acceptable by the stakeholders and the system developers.
- **Requirements Attributes** - provides a repository of a requirement text, with its attributes and requirement traceability relations. It is a very important artifact in the requirements change management.
- **Use-Case Model** - gives the first functional specification overview, giving to developers the first functionality description as models. Contains identified use cases and actors, providing a complete specification of each particular use case. Use-Case Model is shared between many disciplines as a basic document of analysis, design, implementation and testing.
- **Supplementary Specification** - is the complement to the Use-Case Model for the complete specification.
- **Storyboards** - allowing User-Interface Prototype construction
- **Glossary** - which defines a common terminology of concepts used in the project or organization.
- **Software Requirements Specification (SRS)** - includes the complete definition of the system requirements, gathering in one document the use cases models and supplementary specifications. This document is an optional document which may be created on a different scope levels such as a project, a feature or a group of features.

- **Software Requirement** - is a documentation of a capability of the software to solve a problem.

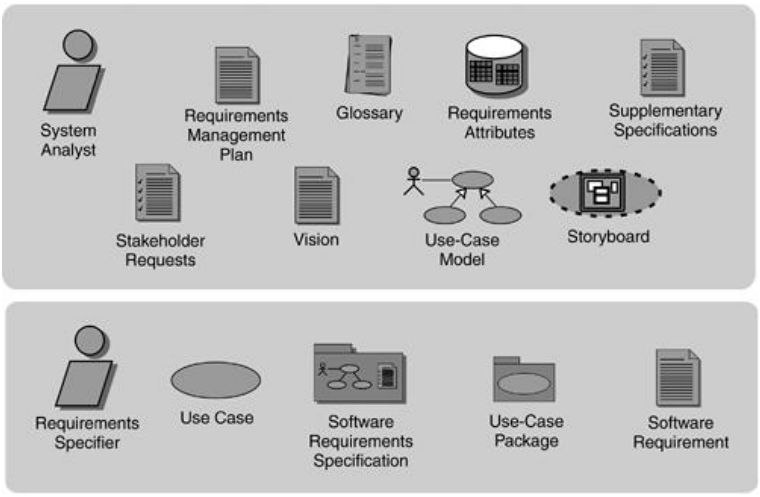

Figure 2.9. Roles and artifacts of the RUP's Requirements discipline

2.5.2.3. Process - activities workflow

In the requirements discipline activities workflow we can distinguish the following actions to be taken:

- **Analyze the problem** - focuses on definition of the project bases by identification of the problem, stakeholders, boundaries of the project and its constraints.
- **Understand stakeholders needs** - requests of various stakeholders and the system's end-users have to be captured and clearly defined for the understanding of the project developers.
- **Define the system** - this action focuses on identification of the system features basing on the information gathered from the stakeholders and users. These features are given a priority and the overall planning of the system's features delivery is made. It also includes the identification of actors and use cases to realize specified features.
- **Manage the scope of the system** - in the domain of the requirements discipline, this action considers taking into account the stakeholders needs in order to manage the requirements definition according to the planned budget and time of delivery of the project. When assigning a priority to a feature its important to be aware of the project scope and available resources and circumstances of its development process.
- **Refine the system definition** - the customer has to accept the requirements captured and specified in a use-case model in order to start the design on the agreed functionalities.

- **Manage changing requirements** - refers to applying some changes to requirements: some new important requirements were discovered, nonfunctional limitation arrised, etc. The agreement with the customer has to be maintained as the requirements change. The customer has to be also aware of the realistic possibilities for delivering the functionalities desired.

The following diagram (Figure 2.10) shows described above activities ordered logically in the requirements discipline workflow.

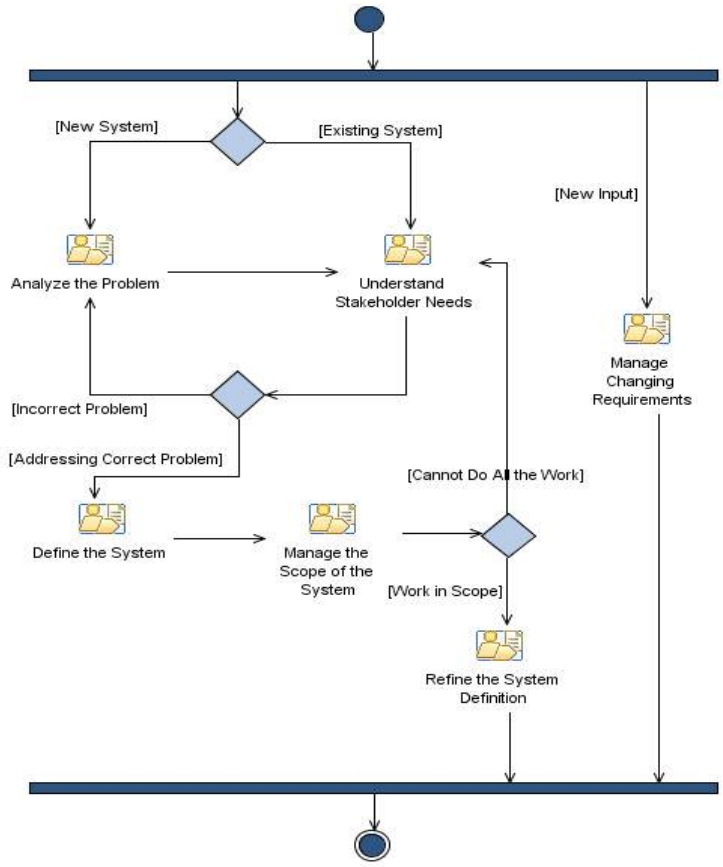

Figure 2.10. RUP requirements discipline workflow

2.5.3. Environment discipline

It is essential to choose a process which fits to the type of product you are developing. Even after a standard process from some existing methodology is chosen, it must not be followed blindly - common sense and experience must be applied to configure the process and tools to meet the needs of the organization and the project. Adapting the RUP methodology and its processes for a project development, taking into account the specific purposes is the key part of the Environment discipline of RUP.

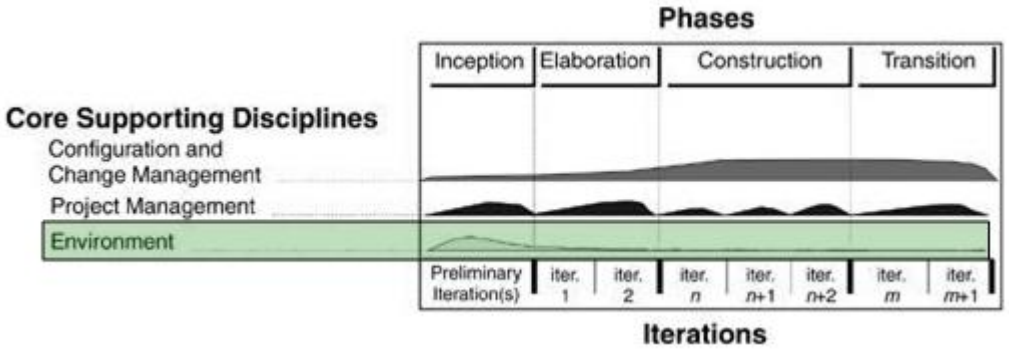

Figure 2.11. Environment Discipline as a RUP core supporting disciplines

The main goal of the environment workflow is to provide adequate support in software development processes, supporting the use of different development tools or methods. A software development organization usually creates a Configuration of the Rational Unified Process, tailored to its context, its need, the size and type of a project. For a specific software development methodologies many steps or activities can be automated in this way facilitating the developer's work and avoiding error-prone aspects of software development processes. This discipline can be also applied to incorporate to one solid development process: aspects, practices or guidelines from different types of architectures or methodologies. The environment discipline consists of the basic process tailoring as well as the process improvment.

The Figure 2.11 presents the RUP supporting disciplines hump chart. A crutial discipline belonging to this group is the Environment discipline, of which a huge part of the work load intensity is condensed in the Inception phase. This section describes in details roles, activities and artifacts that defines this powerfull discipline for development process customization and adaptation.

2.5.3.1. Roles

The RUP defines four roles within the Environment discipline. This section describes these roles and associated to them responsibilities. The Figure 2.12 shows the relation between these roles and artifacts which are responsible for.

- **Process Engineer** - the main goal of this role is to establish an appropriate development process for a project which facilitates the work of developers. For this purpose the Process Engineer tailors the process to specific needs of the organization and project. In addition, the person in this role helps the Project Manager in planning the project. The Process Engineer is resposbible for preparing a series of artifacts, such as: Development Case, Project-Specific Templates, and Project-Specific Guidelines.
- **System Administrator** - the person in this role is responsible for maintaining

the development infrastructure, including hardware and software configuration, required software installation, performing system backup, etc.

- **Tool Specialist** - the main goal of this role is to select and acquire tools, configuring, setting them up, and verify their proper work. Sometimes new tools have to be designed and implemented to automate the process defined by the Process Engineer.
- **Technical Wrtier** - a person in this role is responsible for preparing a technical documentation for end user, such as: user handbook, guides, release notes, etc.

2.5.3.2. Artifacts

The key artifacts of the *Environment* discipline are the following:

- **Development Process** - is a configuration of the underlying RUP framework that meets the specific needs of the project. The process describes the artifacts to be produced as a result of each activity, the roles responsible for them and the timeframe in which they will be produced. It contains such artifacts as: Development Case, Project-Specific Guidelines, and Project-Specific Templates.
- **Development Case** - described the actual process used by the project. The Development Case includes phases and milestones, roles and associated artifacts, guidelines on how to use these artifacts and when they should be produced. It also describes obligatory and optional activities to be performed. Many activities and steps of the Rational Unified Process can be automated through the use of tools, thereby removing the most tedious, human-intensive, and error-prone aspects of software development.
- **Project-Specific Guidelines** - is an artifact which provides guidance on how to perform the tasks defined in the development process. It is an important artifact for maintaining standrads use and producing a quality uniform software.
- **Project-Specific Templates** - defines templates to be used in the project when creating required artifacts. These templates are useful when they are tailored to fit the needs of the particular project.
- **Development Infrastructure** - includes the hardware and software installations.
- **Development Organization Assessment** - describes the current status of the software organization in terms of current process, tools, team members capabilities, etc. It guides the Process Engineer in preparing a process tailored for an organization or a particular project. It is an optional artifact.
- **Manual Styleguide** - supports the uniform creation of user support material to ensure its consistency.

2.5.3.3. Process - activities workflow

The Figure 2.13 shows the *Environment* discipline workflow, which contains the following typical activities:

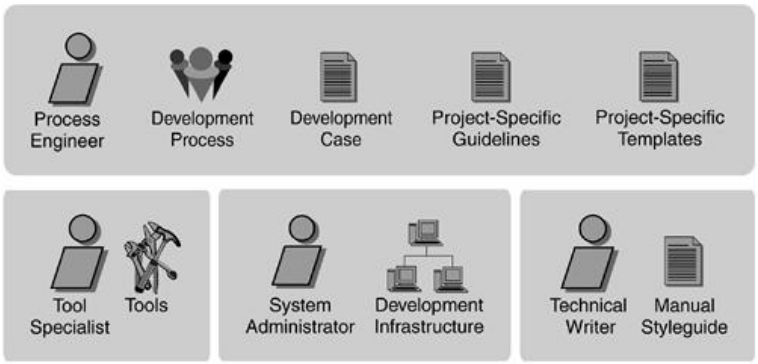

Figure 2.12. Roles and artifacts of the RUP's Environment discipline

- **Prepare environment for project** - an activity of the Inception phase. It ensures that an appropriate process and tools were chosen for the project. The preparation of the environment involves the tailoring of the development process to apply a particular technology, modeling approache, architecture, or any other specific need. The purposes of preparing the environment for a project are:
  - Assess the current development organization.
  - Assess the current tools support.
  - Develop a first draft of the development case.
  - Produce a list of candidate tools to use for development
  - Produce a list of candidate project-specific templates for key artifacts.
  - Prepare Environment for an Iteration
- **Prepare environment for an iteration** - as the RUP defines iterative process, iterations serve to improve the process in further iterations. It leads to refinement of the development environment. This is done by developing or modifying artifacts, such as: Development Case, Manual Styleguides, Project-Specific Guidelines, or Project-Specific Templates. The purposes of preparing the environment for an iteration include the following:
  - Complete the development case to be ready for the iteration.
  - Prepare and if necessary customize tools to use within an iteration.
  - Verify that the tools have been configured and installed correctly.
  - Produce a set of project-specific templates to use within the iteration.
  - Train people to use and understand the tools and the development case.
- **Support environment during an iteration** - an activity that as the main goal has ensuring that an appropriate environment is available for developers to work efficiently and effectively. In involves intalling required software and also solving hardware or network issues.

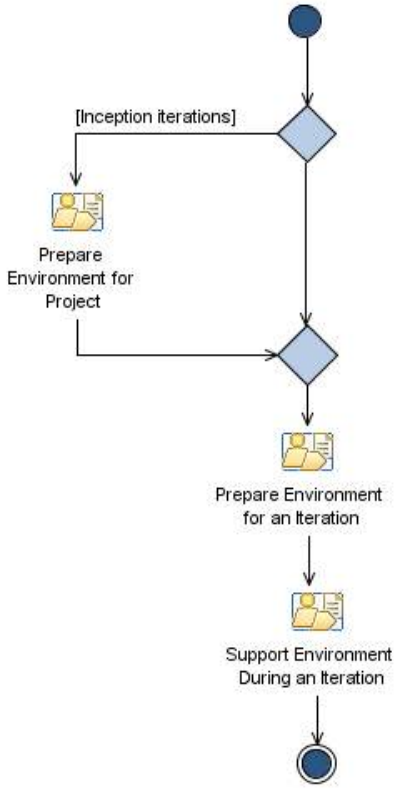

Figure 2.13. Environment discipline workflow in RUP

2.6. OpenUP

Different domains and software projects have different process needs. Depending on the team size, architecture complexity, technology novelty, conformance to standards, etc., more or less formal process is suitable. For this reason a minimally sufficient but complete, agile and extensible development process was developed by the IBM Rational employees. This section is to give an overview of the OpenUP methodology, its origin and definition of its strategic elements such as phases, disciplines, roles, work products, activities and workflows.

2.6.1. Methodology generals

OpenUP methodology, created as a part of the Eclipse Process Framework (EPF) Project, is a simplified version of the RUP. It is minimally sfficient software development process providing only fundamental content for small or medium size projects that deliver software as a main product, but do not require high level of formality. In addition, it suits well to projects where stakeholders are involved in the project significantly and define requirements incrementaly. RUP disciplines such as business modeling, environment or deployment are not included as they were considered as

suitable parts of the development process only for a large projects. Moreover, OpenUP might not be a suitable process for projects which require high level of formality, do not include stakeholders as active roles in the process and deliver other artifacts than software.

OpenUP adapts from RUP some basic elements, such as: definition of the methodology structure (i.e. phases, roles, artifacts, tasks), iterativeness, principles of an incremental and architecture-centric approach. It also considers risk managemenet through the entire process.
OpenUP joins many practicies that help teams to be more effective in developing software, such as: agile philosophy that focuses on the collaborative nature of software development, extensibility which allows the process to be extended or tailored for specific needs of a project or organization through identified reusable content and process elements. It is also tools-agnostic and low-ceremony process that has application in a broad variety of project types.

2.6.2. Main characteristics

OpenUP is driven by four core principles with which to capture the general intentions behind a process and to interpret roles, work products, and tasks. These principles are as follows:

- Collaborate to align interests and share understanding. This principle promotes practices that foster a healthy team environment, enable collaboration and develop a shared understanding of the project.
- Balance competing priorities to maximize stakeholder value. This principle promotes practices that allow project participants and stakeholders to develop a solution that maximizes stakeholder benefits, and is compliant with constraints placed on the project.
- Focus on the architecture early to minimize risks and organize development. This principle promotes practices that allow the team to focus on architecture to minimize risks and organize development.
- Evolve to continuously obtain feedback and improve. This principle promotes practices that allow the team to get early and continuous feedback from stakeholders, and demonstrate incremental value to them.

2.6.3. Process lifecycle

OpenUP is an iterative methodological approach what means that the software development process is split into iterations [ref def]. Typical iteration in OpenUP lasts from 1 week to 6 weeks. Software it this approach is produced incrementally, i.e. the results of one iteration serve as a base for the next iteration.
OpenUP organizes iterations into a set of phases applying the RUP phases: *Inception*, *Elaboration*, *Construction*, and *Transition*. The following Figure 2.14 shows the deliv-

ery process based on the mentioned here phases in a end-to-end software development lifecycle.

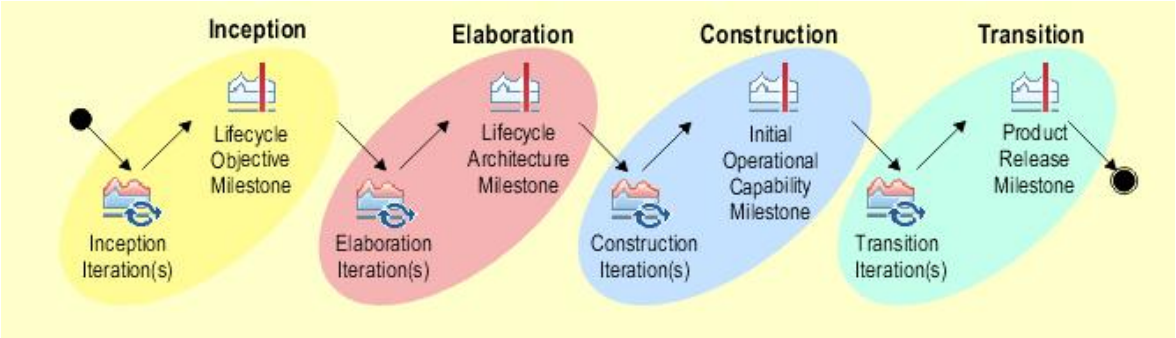

Figure 2.14. OpenUP delivery process *(extracted from the OpenUP methodology)*

2.6.3.1. Inception

The first phase of the development process, where stakeholders and team members collaborate to determine the project scope and objectives. The purpose in this phase is to take the decision whether or not the project should proceed. In this phase the project scope has to be clarified as well as project objectives and the feasibility of the intended solution. After this phase it should be clear who is interested in this system and why, what are the key system functionalities and which of them are the most critical. Also a possible solution should be identified together with a candidate architecture. All mentioned here activities help the project manager to estimate the project cost, plan its schedule and understand the associated with the project risks.

2.6.3.2. Elaboration

The second phase in the project lifecycle, when architecturally significant risks are addressed. The main purpose of the phase is to get a more detailed understanding of the requirements, in particular the critical requirements should be well studied to be calidated by the architecture. This phase should give us an answer to the question about the executable architecture adequacy for developing the application. Also technical and non-technical risks are evaluated. Many technical risks are addressed as a result of detailing the requirements and of designing, implementing, and testing the architecture. Non-technical risks take into consideration the legal and financial issues related to usage of open source or commercial components. Identifying these essential risks alllow producing accurate schedule of a high-level project plan and cost estimations.

2.6.3.3. Construction

This is the third phase of the process, which focuses on detailing requirements, designing, implementing, and testing the bulk of the software. The purpose in this phase is to cost-effectively develop a feature-complete product (an operational version of your system) that can be deployed in the user environment. The product is developed iteratively, where the completeness of the product to be released determines the number of iterations in the phase.

2.6.3.4. Transition

This is the last phase of the process, which focuses on transitioning the software into the customer's environment and achieving stakeholder concurrence that product development is complete and ready for delivery to users.

2.6.4. Disciplines

OpenUP within the method content defines a set of 6 disciplines: *Requirements*, *Architecture*, *Development*, *Test*, *Project Management*, and *Configuration & Change Management*. These disciplines are considered essential, necessary and sufficient for small projects in an agile approach. The Figure 2.15 shows a hump chart presenting all disciplines and the intensity of the work in each of them. Comparing to the RUP, such disciplines as Business Modeling or Environment are considered unnecessary and thus were removed. The purposes for each discipline are listed below.

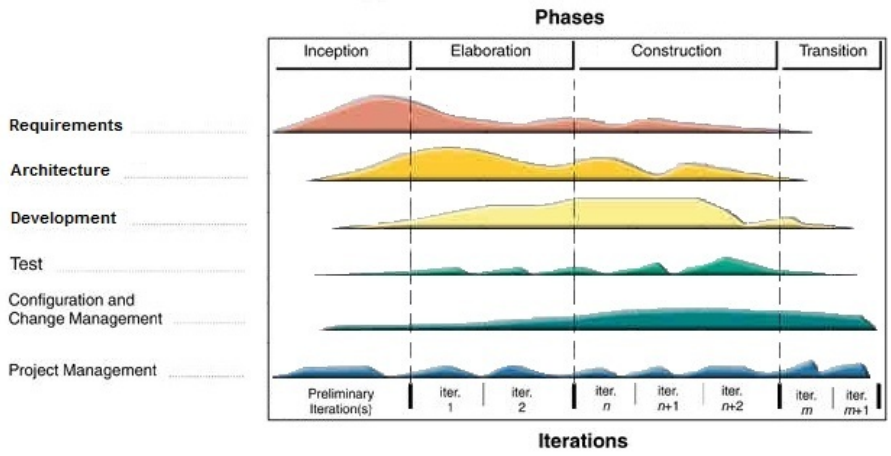

Figure 2.15. OpenUP disciplines hump chart

Requirements

- to understand the problem to be solved;
- to understand stakeholder needs (what users want);
- to define the requirements for the solution (what the system must do);

- to define the boundaries (scope) of the system;
- to identify external interfaces for the system;
- to identify technical constraints on the solution;
- to provide the basis for planning iterations;
- to provide the initial basis for estimating cost and schedule;

Architecture

- to provide guidelines on how to create an architecture from architecturally significant requirements;
- to evolve a robust architecture for the system;
- to define the representation of the architecture (either formal or informal documents);

Development

- to transform the requirements into a design of the system-to-be;
- to adapt the design to match the implementation environment;
- to build the system incrementally;
- to verify that the technical units used to build the system work as specified;

Test

- to provide early and frequent feedback that the system satisfies the requirements;
- to objectively measure progress in small increments;
- to identify issues with the solution;
- to provide assurance that changes to the system do not introduce new defects;
- to improve velocity by facilitating the discovery of issues with requirements, designs, and implementations as early as possible;

Configuration and change management

- to maintain a consistent set of work products as they evolve;
- to maintain consistent builds of the software;
- to provide an efficient means to adapt to changes and issues, and re-plan work accordingly;
- to provide data for measuring progress;

Project management

- to encourage stakeholder consensus on prioritizing the sequence of work;
- to stimulate team collaboration on creating long term and short term plans for the project;
- to focus the team on continually delivering tested software for stakeholder evaluation;

- to help create an effective working environment to maximize team productivity;
- to keep stakeholders and the team informed on project progress;
- to provide a framework to manage project risk and continually adapt to change;

2.6.5. Roles

OpenUP methodology defines 7 different roles, but depending on the project dimensions this number can vary. Division by roles can not only be applied to the activities classification that should be executed by a particular role, but also to define the capabilities that have to own a praticular person. This is also helpful when selecting appropriate project team members. Every role is responsable for creation or changing of a particular artifact. The list below presents the roles defined by the OpenUP methodology.

- **Stakeholder** - represents interest groups whose needs must be satisfied by the project. It is a role that may be played by anyone who is (or potentially will be) materially affected by the outcome of the project.
- **Project Manager** - leads the planning of the project in collaboration with stakeholders and team, coordinates interactions with the stakeholders, and keeps the project team focused on meeting the project objectives.
- **Analyst** - represents customer and end-user concerns by gathering input from stakeholders to understand the problem to be solved and by capturing and setting priorities for requirements.
- **Architect** - is responsible for designing the software architecture, which includes making the key technical decisions that constrain the overall design and implementation of the project.
- **Developer** - is responsible for developing a part of the system, including designing it to fit into the architecture, and then implementing, unit-testing, and integrating the components that are part of the solution.
- **Tester** - is responsible for the core activities of the test effort, such as identifying, defining, implementing, and conducting the necessary tests, as well as logging the outcomes of the testing and analyzing the results.
- **Any Role** - represents anyone on the team that can perform general tasks.

2.6.6. Artifacts

OpenUP defines 17 artifacts which are considered as essential artifacts of the software development. These artifacts, classified by the discipline of application, are the following:

- Architecture
  - Architecture Notebook - describes the context for software development; it contains the decisions, rationale, assumptions, explanations and implications of forming the architecture;

- Requirements
  - Vision - contains the definition of the stakeholders' view of the product to be developed, specified in terms of the stakeholders' key needs and features;
  - Supporting Requirements Specification - captures system-wide requirements not captured in scenarios or use cases, including requirements on quality attributes and global functional requirements;
  - Use Case - captures the sequence of actions a system performs that yields an observable result of value to those interacting with the system;
  - Use Case Model - contains defined use cases emphasizing the relations between them;
  - Glossary - defines important terms used by the project;
- Development
  - Desing - describes the realization of required system functionality in terms of components and serves as an abstraction of the source code;
  - Implementation - represents software code files, data files, and supporting files such as online help files that represent the raw parts of a system that can be built;
  - Developer Test - the instructions that validate individual software components perform as specified;
- Test
  - Test Case - is the specification of a set of test inputs, execution conditions, and expected results, identified for the purpose of making an evaluation of some particular aspect of a scenario;
  - Test Log - collects raw output captured during a unique execution of one or more tests for a single test cycle run;
  - Test Script - contains the step-by-step instructions that realize a test;
- Project management
  - Project Plan - gathers all information required to manage the project; contains a description of project phases and milestones;
  - Iteration Plan - contains plan describing the objectives, work assignments, and evaluation criteria for the iteration;
  - Risk List - is a list of known and open risks to the project, sorted in order of importance and associated with specific mitigation or contingency actions;
  - Work Items List - contains a list of all scheduled work to be done within the project, as well as proposed work that may affect the product in this or future projects; it is frequently used for project estimations;
- Configuration management
  - Build - is an operational version of a system or part of a system that demonstrates a subset of the capabilities to be provided in the final product;

2.7. Methods Engineering

The correct use of software development process is a chalange in many projects where only some generic methods are given for the project team members. The process implementation is no an easy task. The processes are complex, highly iterative, with parallelisms and several relationships among tasks. Moreover, the process itself is frequently not well defined and the environment in which the process to be applied faces the problem of constantly changing business requirements. It is very important to provide a project with easy to understand development process, adaptable for specific organizations or even particular project needs. Methods engineering is a discipline that emerged as a response to the increasing complexity and diversity of information systems developments. It help in situational and domain-specific methods construction as an alternative for the use of ad-hoc methods. This section describes standards and tools that help in effective generic methodologies adaptation, being a solution to a recent common problem of "one method fits all". In particular it provides a detailed description of one architecture of software process engineering - the Unified Method Architecture (UMA), and one supporting tool - Eclipse Process Framework (EPF).

2.7.1. Software Process Engineering

When methodologies first emerged in the 1990s, each software development process used its own concepts and notations to define the contents of the methodology. The need to unify all these concepts and notations therefore emerged. The OMG thus introduced the Software Process Engineering Metamodel (SPEM) [63] standard. SPEM provides a complete metamodel based on the Meta Object Facility (MOF) [62] to formally express and maintain development method content and processes.

In this master thesis, the Unified Method Architecture (UMA) [30] is used for modeling of software process engineering. UMA is an architecture that clearly separates Method Content definitions from their application in delivery processes. UMA is an evolution of SPEM v1.1 [2], but IBM and OMG have worked on UMA to make it part of SPEM 2.0.

UMA defines the schema and terminology used to represent methods consisting of method content and processes. Its basic elements are shown on the Figure 2.16. According to the UMA structure elements defined, UMA is made of reusable core *Method Content* in the form of a general content descriptions and also project-specific process descriptions.

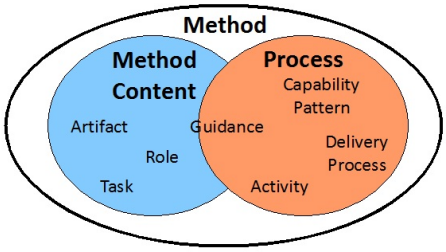

Figure 2.16. The basic elements of UMA

Unified Method Architecture

As depicted on the Figure 2.16, UMA consists of content elements and process elements. The former describe the schema elements for the static aspects of a process, while the latter describe the process organazing specified content elements into activities and workflows. The basic content elements are the following:

- role - is a set of related skills, competencies, and responsibilities; a person in a specific role is responsible for performing associated tasks and producing related work products;
- work product - is a result of a specific task; UMA distinguishes three types of work products: artifacts, outcomes, and deliverables;
- task - is an action performed by roles, associated with input and output work products;
- step - represents the most granular unit of work to be performed; it is a means of breaking down tasks into more atomic work units;
- guidance - is a detailed description of a task, work product, or process element; some types of guidance defined by the UMA are the following: checklist, example, guideline, report, supporting material, etc.;

The basic elements that allow a precise process definition are the following:

- activity - is a process element that boundles method content elements (specifically tasks); it is a work breakdown element that groups a unit of work;
- iteration - allows process engineers to group activities that are planned to be repeated more than once; it is a special form of activity;
- phase - groups process elements into a significant period in a project; phases are not expected to be repeathed as activities or iterations;
- capability pattern - is a reusable process fragment that can contain activities and milestones; a delivery process is frequently a composition of different capability patterns;
- delivery process - represents end-to-end project lifecycle that includes phases, iterations, capability patterns and milestones;

- milestone - is a decision point which compares the products of an activity, an interation or a phase with expected results for these process elements;
- process package - groups process elements into folders for organizational purposes, this feature is optional;

2.7.2. Tool support

The most challenging task of a process engineer is to deploy a methodology in an organization or adapt an existing methodology and processes for a particular project. This task includes configuring and customizing the base methodology.

To address this issue, various tools have been introduced that support aforementioned standards for method engineering. One of these tools is the IBM Rational Method Composer (RMC) [30]. RMC is a UMA-based comprehensive process authoring tool which provides extensive method authoring and publishing capabilities [72]. RMC uses the concept of a plug-in library to allow process engineers to define and extend methodologies.

Another tool that supports the UMA standard for methods engineering is the Eclipse Process Framework (EPF) [2]. In the present work the EPF tool is used to extend OpenUP by incorporating a model-driven requirements engineering approach.

Proposed within this thesis approach is based on the OpenUP methodology, as will be described in Chapter 4. The fact that OpenUP is itself a plug-in library of the EPF tool, it permits to define new processes or extend the existing one based on the OpenUP.

EPF Composer has two main purposes, which are the following:

- To provide for development practitioners a knowledge base that allows them to browse, manage, and deploy content. This content can accommodates your own content consisting of method definitions, whitepapers, guide-lines, templates, principles, best practices, internal procedures and regulations, training material, and any other general descriptions of how to develop software. This knowledge forms the basis for developing processes.
- To provide process engineering capabilities by supporting process engineers and project managers in selecting, tailoring, and rapidly assembling processes for their concrete development projects. EPF Composer provides catalogs of pre-defined processes for typical project situations that can be adapted to individual needs. It also provides process building blocks called capability patterns that represent best development practices for specific disciplines, technologies, or development styles. EPF Composer also allows you to set-up your own organization-specific capability pattern libraries. Finally, the documented processes created with EPF Composer can be published and deployed as Web sites.

Chapter 3

Related work

This chapter presents studies realized with the objective to gather current knowledge about the requirements engineering methods and techniques that are being used in the context of the model-driven development. Also requirements engineering tasks within the health-care domain were investigated, as well as the SOA-based methodological approaches. This knowledge served to define the objectives of the present master thesis.

Section 3.1 presents a research on the RE techniques in the model-driven development processes and their actual level of automation. In order to investigate this issue in greater depth, a systematic literature review has been performed. Also in this section a well-defined method for systematic revisions is briefly discussed describing the steps performed in the revision process. Related approaches were analyzed from different points of view, such as: requirement types used, requirements structure, use of transformations and their automation, possibility of requirements traceability and its automation, tool support provided, or validation method. In addition, an analysis of the principal findings and this systematic revision limitations are described.

Section 3.2 includes a brief analysis of the principal commonly known approaches to service-oriented systems development. A short characteristic of each one is provided focusing on requirements modeling in these approaches.

Section 3.4 contains a nutshell description of the requirements engineering processes in the health-care domain. A short summary of found in the literature approaches are discussed, it being a motivation for the methodological proposal developed within this master thesis.

The last section of this chapter concludes the existing research on the requirements engineering techniques for the SOA-based health-care systems in the model-driven context described in the literature.

3.1. Requirements Engineering in Model-Driven Development

Software engineering experiences show that in recent decades model-based development of systems has become an essential factor in reducing costs and development time. Furthermore, properly managed, well-documented and easily understandable software requirements definitions have a great impact on final product quality [16]. However, requirements engineering (RE) is one of the software engineering disciplines in which model-based approaches are still not widespread. Requirements are generally regarded as text fragments that are structured to a greater or lesser extent and which are interpreted by stakeholders and developers, who manually manage the requirement interrelationships [50].

A variety of methods and model-driven techniques have been published in literature. However, only a few of them explicitly include the requirements discipline in the Model-Driven Development (MDD) process. This section presents a review of scientific papers published in the last decade which include the use of RE techniques in the context of an MDD process. In order to provide a balanced and objective summary of research evidence, a systematic literature review (SLR) process is considered as an appropriate method to carry out such a review in software engineering [10].

The section contains a description of the protocol followed in this review, presents the results obtained and a discussion on the threats to the validity of the results. Finally, some conclusions on the requirements engineering techniques applied in the model-driven context are drawn. This section outlines the issue for researchers, which is how to improve the current practices of the use of MDD techniques at the requirements stage.

3.1.1. Research method

The approach proposed by Kitchenham [38] has been followed for systematic literature review. A systematic review is a means of evaluating and interpreting all the available research that is relevant to a particular research question, topic area, or phenomenon of interest. It aims at presenting a fair evaluation of a research topic by using a trustworthy, rigorous, and auditable methodology. A systematic review involves several stages and activities (see Figure 3.1), which are briefly explained below.

- **Planning the review**: the need for the review is identified, the research questions are specified and the review protocol is defined.
- **Conducting the review**: the primary studies are selected, the quality assessment used to include studies is defined, the data extraction and monitoring is performed and the obtained data is synthesized. In this stage we added a new activity to test the reliability of the review protocol.
- **Reporting the review**: dissemination mechanisms are specified and a review

report is presented.

The activities concerning the planning and conducting phases of this systematic review are described in the next sections.

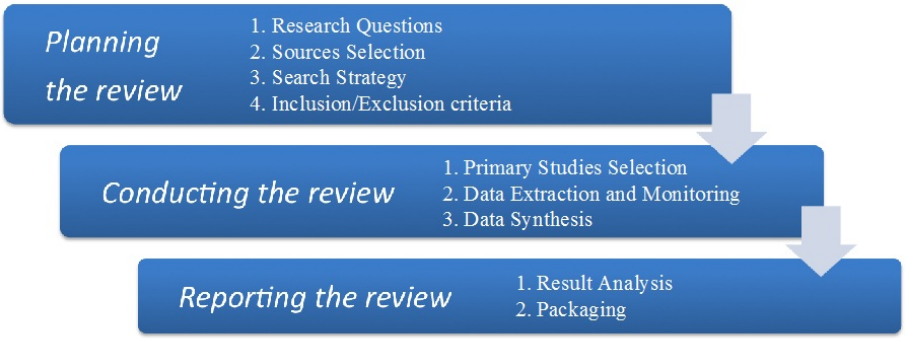

Figure 3.1. Phases and activities of the systematic literature review

3.1.1.1. Research question

According to the methodology for systematic reviews, the first step is to establish the research question. In order to examine the current use of requirements engineering techniques in model-driven development and their actual level of automation, we formulated the following research question: "What requirements engineering techniques have been employed in model-driven development approaches and what is their actual level of automation?". The intention of this research question was to enable us to define a process with which to collect current knowledge about requirements engineering techniques in MDD and to identify gaps in research in order to suggest areas for further investigation. The review has been structured by following the PICOC criteria [65]:

- **Population:** Research papers presenting MDD processes and techniques,
- **Intervention**: Requirements engineering methods and techniques,
- **Comparison**: Analysis of all approaches based on the specified criteria,
- **Outcome**: Not focused on achieving any specific result,
- **Context**: Research papers based on RE techniques used in MDD.

Our review is more limited than a full systematic review, such as that suggested in [38], since we did not follow up the references in papers. In addition, we did not include other references such as technical reports, working papers and PhD thesis documents. This strategy has been used in another systematic review conducted in the software engineering field [55].

3.1.1.2. Sources selection

Two types of search methods were used to select appropriate and representative papers in the field of requirements and model-driven engineering. The first type, automatic searching, was based on four main sources of scientific paper databases: IEEE *Xplore* (IE), ACM Digital Library (ACM), Science Direct (SD), and SpringerLink (SL). A manual search was also carried out in the following representative conferences and journals: Requirements Engineering conference (RE), the Conference on Model Driven Engineering Languages and Systems (MODELS), and Requirements Engineering Journal (REJ).

This manual search method was applied in order to verify the correctness of the automatic review and to carry out a more in-depth study of those works published in these sources that explore new trends and approaches.

3.1.1.3. Identifying and selecting primary studies

The research goal, was used as a basis for the creation of a search string with which to identify primary studies. The search string consisted of three parts: the first part linked those works that describe requirements engineering techniques using models, the second part was related to model-driven engineering concepts, and finally the third part described the relation between requirements and other models in the MDD process. We experimented with several search strings and the following retrieved the greatest amount of relevant papers:

(requirements engineering **OR** *requirements-based* **OR** *requirements-driven* **OR** *requirements model* **OR** *business model* **OR** *CIM* **OR** *Computation Independent Model)*
AND
(MDA **OR** *MDE* **OR** *model-driven* **OR** *model-based* **OR** *model*)*
AND
*(transform** **OR** *traceability)*

The concrete syntax of this search string was adapted to each digital library we used.

3.1.1.4. Inclusion criteria and procedures

The searching configuration included limitations to the type of papers and content. Papers that had been taken into consideration were only those that are research papers presenting approaches to MDD-based requirements engineering or software development process with requirements traceability. Moreover, only papers published in conferences/workshops proceedings and scientific journals between January, 1998 and January, 2010 were considered as significant for the research. The following types of papers were excluded:

- papers describing model-driven principles without describing a concrete requirements engineering technique,

- papers presenting requirements engineering techniques that are not related to model-driven principles,
- books, tutorials, standards definitions, poster publications,
- short papers (papers with less than 4 pages),
- papers not written in English.

3.1.1.5. Data extraction strategy

The data extracted were compared according to the research question stated, which is here decomposed into more specific questions and resulting in establishing criteria that are described further in this section. These questions are presented in the following list, where the possible answers are given in the round brackets.

1. What is the type of level for requirements specification? (Software, Business)
2. How the requirements specification is organized/structured? (Models or Diagrams, Templates, Structured natural language, Natural language, Other)
   If models are provided, are they specified using standards? (Yes, No)
3. If models are used to represent the requirements, what is the type of these models? (Structural, Behavioral, Functional, Other)
4. Does the approach provide model transformations from the Requirements Engineering phase to the analysis and/or design phases? (Yes, No)
5. What is the type of requirements transformations regarding the abstraction level of source and target models? (Endogenous, Exogenous)
6. If model transformations are provided, are these transformations specified using standard languages? (Yes, No)
7. What is the level of automation of these transformations? (Automatic, Interactive, Manual)
8. Is there backwards requirements traceability information? (Yes, No)
   If yes, which phase product can be traced to requirements? (Analysis, Design, Implementation)
9. If traceability is provided, what is the level of automation of the traceability? (Automatic, Manual)
10. Does the approach have a tool support? (Yes, No)
    If yes, what is the purpose of such tool? (Traceability only, Transformations only, Transformations and traceability)
11. Was the requirements engineering technique validated? (Yes, No)
    If yes, what type of validation was conducted? (Survey, Case study, Experiment)
12. What is the actual usage of the approach? (Academic, Industry)

On the basis of these questions, the following criteria has been defined:

- **Type of requirements** (criterion 1). This can be of two types: *software* requirements which are requirements that describe only the functionalities of software under development, and *business* requirements which include information that is related not only to the functionality of the future system, but also to the business context, organizational structure of the enterprise, processes, etc. which will not necessarily be a part of the system to be developed.
- The information concerning the type of **requirements structure** (criterion 2) is then collected. Requirements can be represented as *models* (two types of models are distinguished: *standard* models expressed in the only modeling language that is considered as a standard (UML from OMG) and other *non-standard* types of models). Requirements can also be expressed in *natural language* or other types of textual or graphical representation.
- In the case of using models for requirements specification, the information concerning the **type of models** (criterion 3) is gathered. These models can be: *structural*, *behavioral*, *functional* or of *another* type.
- Model **transformations provided** (criterion 4). This is an interesting topic which concentrates on an important amount of research work.
- **Level of transformations** (criterion 5), as proposed in Mens *et al.* [56]. For transformations we can also analyze the languages in which both source and target models and their abstraction levels are expressed. In this case transformations are classified as *endogenous* when the source and target model are expressed in the same language and in the same abstraction level, and *exogenous* when different modeling languages and abstraction levels are used to express source and target models (e.g. in UML language: a source model can be expressed as a use case model and the target model as an activity diagram).
- **Use of transformation languages** (criterion 6) is also analyzed. Transformations can be defined with *standard* languages such as QVT or ATL[1] or with *non-standard* transformation languages.
- **Transformation automation level** (criterion 7). We consider a transformation to be *automatic* if the entire process of obtaining the target model can be carried out without the transformation user's participation. We then distinguish *interactive* (semi-automatic) or *manual* approaches.
- **Requirements traceability** (criterion 8). Requirements traceability refers to the ability to describe and follow the life of a requirement, in both a forwards and backwards direction. We focus on the *post-requirements specification* (post-RS). Post-RS refers to those aspects of a requirement's life that result from inclusion in a requirements specification as defined by Gotel and Finkelstein [29]. The papers

1. ATL is included in the standard category since it is widely used in academia and can be considered as a *de facto* standard.

reviewed were analyzed to study the traceability to and from *analysis*, *design*, and *implementation* artifacts.

- **Traceability automation** (criterion 9). This is investigated to provide us with some conclusions regarding the effort needed to manage the traceability within the MDD process.
- **Tool support** (criterion 10). We analyzed whether there is a tool that performs the MD transformations on requirements and generates model(s), and also whether it provides support for requirements traceability and its monitoring during the entire software development process.
- **Type of validation** (criterion 11) conducted. Three different types of strategies can be carried out depending on the purpose of the validation and the conditions for empirical investigation [26]: *survey*, *case study*, and *experiment*. A survey is an investigation performed in retrospect, when the method has been in use for a certain period of time. A case study is an observational study in which data is collected for a specific purpose throughout the study. An experiment is a formal, rigorous, controlled investigation. The set of validation methods does not include theoretical examples of proof-of-concepts.
- Finally, the **actual usage** (criterion 12) of the requirements engineering technique is analyzed. The paper is classified as being *industrial* if it presents a requirements engineering technique which was proposed for (and is being used in) the context of a company. Otherwise, it is classified as *academic* if the paper describes an academic environment and no evidence of its current usage is provided.

3.1.1.6. Conducting the review

The stage in which the review is conducted consists of the following activities: selection of primary studies, data extraction, and data synthesis. The previous sections describe how and from which sources the primary studies were selected. Based on this information, the automatic search based on the search string performed in selected digital bibliographic libraries resulted in 867 potentially related papers which might be significant for the research topic.
In addition, a manual search (MS) was also conducted in order to find any relevant papers that might exist and had not been discovered with the automatic search (AS). The MS sources include the RE and MODELS conferences and the REJ journal, and resulted in 17 possibly relevant papers related to the research topic which are additional to those selected by the AS.

Table 3.1 presents the results of the final set of relevant papers selected for each of the sources. *Search results* row shows the number of papers obtained from each source that resulted from the search string and the manual search, the *finally selected* row indicates the number of papers that remained for review after the rejection of papers that satisfied at least one of the exclusion criteria or whose topic was not suitable for

Table 3.1. Number of the review results

	Automatic search				Manual search (MS)	
Source	IE	ACM	SD	SL	RE/MODELS/REJ	Total
Search results	163	641	24	39	17	884
Finally selected	21	25	9	7	10	72

the purpose of the systematic review. Duplicated papers were discarded by taking into consideration the first digital library in which they appear, along with the most recent or the most complete publication.

72 papers were consequently chosen for the final review. The research was first carried out in December 2009 and was then updated by January, 31, 2010. A complete list of the papers reviewed can be found in Appendix D (also available at: `www.dsic.upv.es/~einsfran/review-remdd.htm`).

3.1.2. Results

This section discusses the results of the review, in which each criterion is commented on. Table 3.2 shows a summary of the number of papers obtained as a result of the review. This table is organized into groups regarding the selection criteria and the publication sources.

The results for the **requirements type** (criterion 1) show that the majority of works (60%) focus on *software requirements* (e.g., Insfran *et al.* in [33], in which requirements are represented through the use of the Techniques for Requirements and Architecture Design (TRADE) such as mission statement, function refinement tree, and use cases). In this context, the Service Oriented Architecture (SOA) has gained a significant amount of popularity in recent years. Various works describe automation methods for services specification, e.g., Jamshidi [34] proposes a new method called ASSM (Automated Service Specification Method) with which to automatically specify the architecturally significant elements of the service model work product from the requirements. Only 40% of the papers reviewed use *business requirements* as a basis for further development. At this point it is worth mentioning that many approaches use the i* notation to describe these business requirements. For example, Mazón *et al.* [52] introduce the use of the i* framework to define goal models for data warehouses and automatic conceptual modeling. Other approaches for generating UML models from business requirements also exist, such as that of Raj *et al.* [66]. This approach presents an automated transformation of business designs from SBVR Business Design (Semantics of Business Vocabulary and Rules) to UML models which bridges the gap between business people and IT people. These results are shown in Figure 3.2.

Table 3.2. Systematic review results

Selection criteria		Sources					Total	%
		IE	ACM	SD	SL	MS		
1 Requirements type	Software	14	13	5	6	5	43	60
	Business	7	12	3	2	5	29	40
2 Requirements structure	Standard model	5	9	4	1	6	25	32
	Non-standard model	9	9	2	1	4	25	32
	Template	0	1	0	0	0	1	1
	Structured natural language	6	7	2	0	2	17	22
	Natural language	2	2	1	3	0	8	10
	Other	0	0	1	1	0	2	3
3 Type of models	Structural	3	4	1	1	0	9	15
	Behavioral	8	15	5	2	9	39	69
	Functional	2	2	0	0	2	6	10
	Other	2	1	2	0	0	5	8
4 Transformations provided	Yes	19	19	8	4	8	58	81
	No	2	6	1	3	2	14	19
5 Transformations level	Endogenous	4	4	0	1	0	9	15
	Exogenous	16	17	8	3	8	52	85
6 Standard transformations	Yes	1	4	0	0	2	7	13
	No	16	14	6	4	6	46	87
7 Transformations automation	Automatic	7	11	3	0	6	27	45
	Interactive	4	3	2	1	1	11	18
	Manual	9	8	2	2	1	22	37
8 Traceability requirements	To analysis	8	4	2	5	1	20	24
	To design	3	7	1	1	0	12	14
	To implementation	4	6	1	0	1	12	14
	None	12	14	5	2	8	41	48
9 Traceability automation	Automatic	5	5	3	2	2	17	59
	Manual	4	6	0	2	0	12	41
10 Tool support	Traceability only	1	3	1	3	1	9	13
	Transformation only	5	6	4	2	6	23	32
	Traceability&transformation	1	0	1	0	0	2	3
	None	14	15	3	2	3	37	52
11 Type of validation	Survey	0	0	0	0	0	0	0
	Case study	11	10	2	2	7	32	45
	Experiment	0	1	0	2	0	3	4
	None	10	14	7	3	3	37	51
12 Approach scope	Academic	14	19	7	5	8	53	73
	Industry	7	6	2	3	2	20	27

The results for the **requirements structure** (criterion 2) show that of the papers reviewed, 64% of those that apply some RE techniques in the MDD approach used models as a means to represent the requirements. The two types of models are distinguished as follows: UML *standard models* (32%) (the most frequently used are class, use cases, activity, and sequence diagrams) and *non-standard models* (32%) such as goal, aspect, feature, or task-based requirements models. Other alternatives with which to represent the requirements are: i) *structured natural language* (22%) in which requirements are described in an easy to analyze manner. For example, Mauco *et al.* [51] use a natural language oriented model which models the vocabulary of a domain by means of the Language Extended Lexicon (LEL); ii) *natural language* (10%), for example, Fliedl *et al.* in [27] in which the use of sophisticated tagging in the requirements lexical analysis

Figure 3.2. Results for criterion 1 (type of requirements)

is proposed; iii) *templates* (1%) (e.g. the requirements traceability approach presented by Cleland-Huang *et al.* in [14]) and finally, iv) *other* types of specifications which are mostly proprietary domain specific (3%). These results are shown in Figure 3.3.

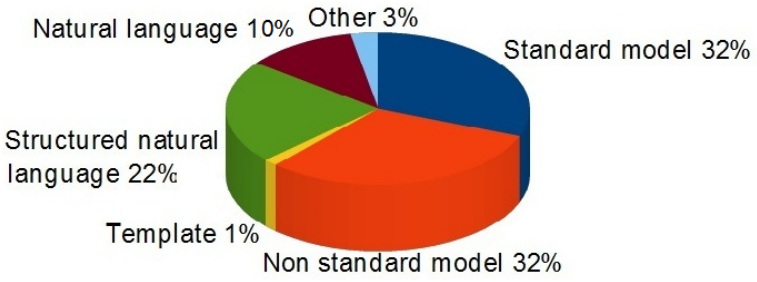

Figure 3.3. Results for criterion 2 (requirements structure)

The results for the **type of models** (criterion 3) show that of those approaches that use models for requirements specification the most frequently used type of model is the *behavioral* model (69%). In many works, this type of model is used as use case specifications (e.g. [45], [31] and [69]) or, very often, as goal models (e.g. in [44], [43] or [42]). Other less frequently used alternatives are: *structural* (15%), *functional* (10%) (e.g. activity or sequence UML diagrams), and *other* types of models (8%) such as the Requirements-Design Coupling (RDC) model proposed by Ozkaya *et al.* [64] or the Stakeholder Quality Requirement Models described with semantic web technology in [8]. A summary of these results is shown in Figure 3.4.

The results for the **transformations from requirements phase** (criterion 4) show that a total of 81% of the papers reviewed describe different types of transformations from requirements specifications. We can distinguish different types of transformations such as mappings, transformations using patterns and templates, transformations implemented in transformation languages (QVT, ATL), linguistic operations on textual requirements or graph transformations, etc. On the other hand 19% of papers do not provide such transformations (see Figure 3.5(A)), and the approach is focused on other aspects of MDD such as the traceability of requirements, e.g. Cleland-Huang *et al.* [14].

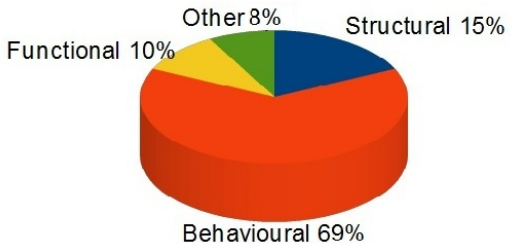

Figure 3.4. Results for criterion 3 (type of models)

The results for the **level of model transformations** (criterion 5) give an outcome concerning the abstraction level of source and target models in the transformation process (see Figure 3.5(B)) according to the aforementioned classification presented by Mens *et al.* in [56]. The vast majority of approaches (85%) transform a source model into a different type of model (*exogenous* transformations). The alteration of the target model specification language or abstraction level in relation to the source models (principally goal models or natural language scenario descriptions) mostly takes place in works that apply UML models as a target model. For example, in [17], Debnath *et al.* describe a natural language requirements transformation to a UML class model. Also, many approaches that start the MDD process from business requirements specifications propose exogenous transformations, as can be seen in [66] in which the business requirements that are specified with the Semantics of Business Vocabulary and Rules (SBVR) are transformed into UML models (other examples might be [37] and [76]). Some works provide transformations of models within the same modeling language, but in a different abstraction level, e.g. transforming UML use case diagrams into UML activity diagrams. *Endogenous* transformations are applied in 15% of the approaches reviewed. For example, this type of transformations is used by Laguna and Gonzalez-Baixauli in [42], in which endogenous transformations are considered as requirements configurations used to validate the completeness and consistency of the initial requirements configurations represented as a set of goal and feature models. Another approach with this kind of transformations was used in a validation of scenarios presented by Shin *et al.* in [71].

The results for the **use of transformation languages** (criterion 6) show that 87% of transformations included in this systematic review use different kinds of specific mappings, refinements or pattern based transformations or languages other than standardized transformation languages (see Figure 3.5(C)), e.g. Raj *et al.* [66] define some specific mapping rules with which to transform business design to UML models. Only 13% of the works use one of the two considered in this work as standard languages: QVT and ATL. For example, in [40] Koch *et al.* propose transformations based on QVT as a transformation language for Web system design.

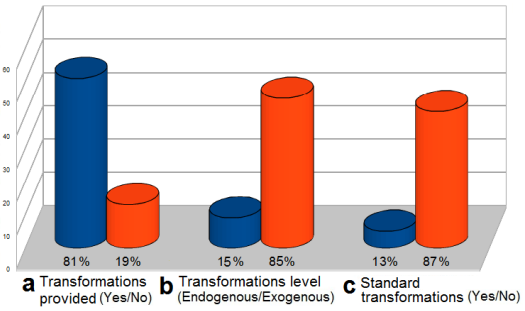

Figure 3.5. Results for criterion 6 (standard transformations)

The results for the **transformations automation level** (criterion 7) show the current state of automation for the transformations defined in the MDD process. 45% of the approaches perform fully *automatic* transitions from requirements specifications to analysis and design models. For example, Zhang and Jiang in [76] propose a well-defined mapping of requirements defined in the Visual Process Modeling language (VPML) at the CIM level to the Business Process Execution Language (BPEL) description at the PIM level. 18% of the papers describe *interactive* or semi-automatic transformation methods, e.g. [43] or [48]; 37% of the papers discuss *manual* transformations, e.g. [42]. Figure 3.6 shows a summary of the results for this criterion.

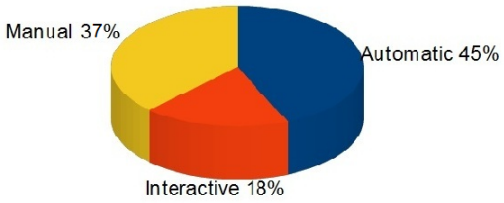

Figure 3.6. Results for criterion 7 (transformations automation)

The results for the **traceability requirements** (criterion 8) show support for requirements traceability in the papers reviewed. With regard to the classification of traceability presented in [29], this work focuses on post-RS traceability, which is the relationship between requirements specification (RS) artifacts and analysis, along with design artifacts. This traceability can be forward and backward traceability. Since this work deals with the model-driven environment, the majority of approaches that possess model transformations assume that forward traceability exists, although not all of these approaches have explicit mechanisms for that traceability support. 48% of works do not provide backwards traceability, although forward traceability is possible. This situation arises in the approach described by Insfran in [33], in which the forward traceability is implicitly assumed by the transformation rules provided, yet there is no reference to the backward traceability. Moreover, 24% of works provide backward traceability from

the analysis phase (e.g. in [43], in which goal models are transformed into statecharts and the backward traceability is recorded); 14% of works provide traceability from design and implementation phases mainly from user interface prototypes and test case implementations (e.g., Sousa *et al.* in [73] present an approach for requirements tracing from user interface implementation). In addition, some authors, such as Naslavsky *et al.* in [59], describe complete solutions for tracing products from different phases to requirements (specified as scenarios). These results are shown in Figure 3.7.

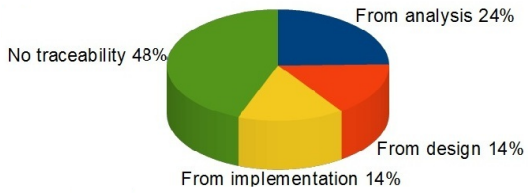

Figure 3.7. Results for criterion 8 (requirements traceability)

The results for the **traceability automation** (criterion 9) show that more than half of the methods (59%) that have some traceability support provide an automated tool for traceability management, e.g. [59]. This signifies that in these approaches the effort needed to manage the requirements traces within the MDD process is quite low, or is none-existent. The number of manual traceability approaches is also significant: 41%. For example, in the work of Sousa *et al.* [73] user interfaces can be logically linked to the business processes defined at the requirements level. These results are shown in Figure 3.8.

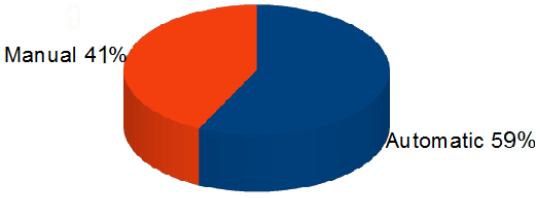

Figure 3.8. Results for criterion 9 (traceability automation)

The results for the **tool support** (criterion 10) for the MDD approach show that of the papers reviewed, as was expected, not even half of them have tool support. With regard to those approaches that have some type of process automation tool, the following categories are distinguished: 32% of approaches have tool support for model transformations, e.g. [76], in which a tool for automatic BPEL models creation is supported based on the source model; 13% of works only support traceability, e.g., in [14] Cleland-Huang *et al.* propose a traceability of requirements specification without

any previous model transformations; and finally, only 3% of the papers describe technological solutions including both transformations and traceability support. One of the works of Rash *et al.* [68] could serve as a good example here since it provides R2D2C transormations in addition to including traceability support. Retrieving models and formal specifications from existing code is also possible.

On the other hand, 52% of works do not offer any tool support. However, most of them emphasize this necessity and state that it will be a part of their future work. These results are shown in Figure 3.9.

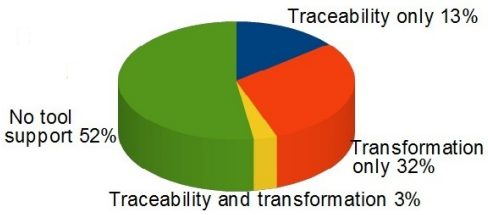

Figure 3.9. Results for criterion 10 (tool support)

The results for the **type of validation** (criterion 11) give an overview of the evaluation methods used in the papers selected. Three validation methods were considered in order to classify the results: survey, case study, and experiment. 52% of the papers reviewed do not present any sort of validation, and a more or less detailed example is the only mean provided to ilustrate the feasibility of the approach. More or less well-defined case studies were used in 44% of the cases. The majority of the papers, particularly those describing academic research, use theoretical examples (e.g. [46]), whereas industrial research, were very often evaluated with a *case study* (e.g. [70] where analysis models (use cases) are generated from textual requirements of a Mobile Media system), although this also ocurred in the other types of research. It is also worth noting that controlled *experiments* were used in only 4% of the works (e.g. [71] or [39]), and validation via *surveys* were never used in the reviewed papers. These results are shown in Figure 3.10.

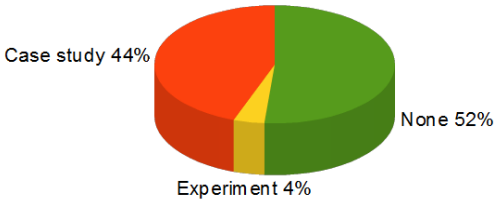

Figure 3.10. Results for criterion 11 (type of validation)

Finally, the results for the **actual usage** (criterion 12) show that 73% of the selected papers were defined in an *academic* context and 27% were defined in an *industrial* setting. The predominance of the academic proposals found in this review shows that new approaches to deal with techniques for modeling, transformations, and processes, that include RE as a part of the MDD process in industrial contexts, are still needed. Some representative attempts from industry to apply model-driven RE in the development process is the AQUA project [24] and [9], where Boulanger *et al.* describe the use of the automotive architecture description (EAST-ADL) and SysML for requirements modeling, traceability and transformation. These results are shown in Figure 3.11.

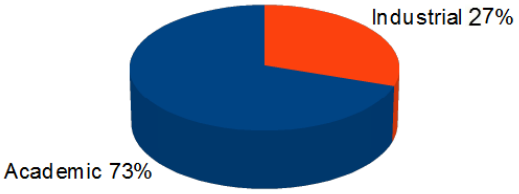

Figure 3.11. Results for criterion 12 (actual usage)

3.1.3. Threats to validity

This section discusses the threats to validity that might have affected the results of the systematic review. The review protocol used has been validated to ensure that the research was as correct, complete, and objective as possible. With regard to the source selection I have selected four digital libraries (IEEEXplore, ACM, Science Direct, and Springerlink) containing a very large set of publications in the Software Engineering field. The search string was defined by attempting different combinations of terms extracted from papers concerning Requirements Engineering and model-driven techniques. Also patterns were applied for search terms and adapted the search string to advanced methods of source searching in each digital library selected. This made the reproducibility of the automatic search for results possible.

Possible limitations of this study concern publication bias, publication selection, inaccuracy in data extraction, and misclassification. Publication bias refers to the problem that positive results are more likely to be published than negative results [38]. I have attempted to alleviate this threat, at least to some extent, by scanning relevant conference proceedings and journals. In addition, the digital libraries contain many relevant journal articles and conference proceedings. With regard to publication selection, the sources in which RE and MDE works were normally published has been chosen. However, the review did not consider grey literature (e.g. industrial reports, PhD thesis) or unpublished results.

With regard to the search string, we attempted to collect all the strings that are representative of the research question. The search string was refined several times in order to obtain the maximum number of papers related to the systematic review. Also synonyms were considered and have been included the lexeme of words. To alleviate the threats to inaccuracy in data extraction and misclassification by conducting the classifications of the papers was done by two reviewers. The discrepancies among the evaluations were discussed and a consensus was reached.

3.1.4. SLR conclusions

Research in the last decade has shown increasing progress with regard to the precision and automatic support that can be applied to requirements engineering specifications. However, a complete solution which includes requirements models as part of MDD processes is still lacking. In addition, little tool support with which to manage requirements models and to make further use of them in an automatic manner is currently provided. Many MDD methodologies include some requirements engineering activities but these are hardly ever included in the mainstream of automated model transformation and code generation activities. Moreover, this systematic review verifies that models are not exclusively used to describe requirements in the MDD context to serve as the input for model-driven transformations, but that templates, structured and non-structured natural language are also used. To date, automated model transformations appear to be poorly used since the majority of them are designed as different kinds of mappings and graph transformations that use non-standard transformation languages.

After analyzing the results of our systematic review we can draw the conclusion that models are not as frequently used as we expected in the requirements phase of a MDD envirnoment (only 64%). Natural language specifications are also very important. Furthermore, post requirements specification traceability is not well-defined. In addition, there are no complete proposals that are well-documented and evaluated, particularly in the industrial setting. Moreover, there is a lack of more empirical studies in MDD environments to show the benefits of the use of RE techniques as a part of the automated transformation processes.

3.2. SOA-focused methodological approaches

3.2.1. SOA-focused methodologies

Service-Oriented Architecture is a powerful software engineering approach which however, demands support of many underlying technologies, such as: WS, BPEL, UDDI, WSDL. All these technologies are important and necessary to achieve SOA and get

expected benefits, such as business alignment, components flexibility, loose coupling, and reusability. However, these technologies are not sufficient on their own without a well-defined end-to-end service delivery methodology and a set of supporting the methodology tools. Such a systematic and comprehensive approach is of critical importance in SOA.
To address this issue, a variety of modeling techniques and methodological approaches for service-oriented development process guidance have been published in literature. However, with regard to the main concern of this master thesis, only a few of these approaches focus on the requirements specifications and aim to automate the generation of the analysis (PIM-level) models from the requirements (CIM-level) artifacts. [47].

This section, describing various service-oriented methodological approaches, is partially based on a state-of-the-art survey presented by Ramollari *et al.* in [67]. It is a comparative study of actual approaches and methodologies of service-oriented development which defines several criteria and characteristics that are used for the methodologies comparison.

IBM Service-Oriented Analysis and Design (SOAD) [77]
SOAD is an abstract framework proposing elements that should be part of a service-oriented analysis and design methodology. SOAD builds upon existing, proven techniques, such as OOAD, CBD, and BPM. It also introduces SOA-specific techniques, such as service conceptualization, service categorization, aggregation, discovery, policies and aspects.

IBM Service Oriented Modeling and Architecture (SOMA) [4]
In the service-oriented modeling context, IBM offers a well-defined approach called SOMA which is a method for developing a service-oriented solutions. SOMA takes a top-down approach to analyze business domains and decompose them in to business processes, sub-processes and use cases, and also a bottom up approach to services architecture. It provides the software engineers with a set of well-defined activities and work products to model functional requirements as services. It consists of three steps: identification, specification, and realization of services, flows (business processes), and components realizing services. The process is highly iterative and incremental. However, because SOMA is proprietary to IBM, its full specification is not available.

Service Oriented Unified Process (SOUP) [58]
Owing to the extensibility and adaptability capabilities of Rational Unified Process (RUP), few approaches use this methodology as a base. One of these methodologies is SOUP [58]. SOUP is a lightweight instance of RUP with SOA specific deliverables for business services and for business processes models and rules. However, this methodology is rather model-based than model-driven. It describes the requirements elicitation and modeling techniques for service-oriented development, but does not provide any

information on the transformation techniques between models. This approach by K. Mittal consists of six phases: incept, define, design, construct, deploy, and support. However, SOUP lacks detailed documentation and leaves room for adaptation. It is used in two slightly different variations: one adopting RUP for initial SOA development and the other adopting a mix of RUP and XP for the maintenance of existing SOA projects.

MINERVA framework [18]
Delgado *et al.* [18] introduce MINERVA framework which applies model-based development and SOC paradigms to service-oriented development methodology. This approach is RUP independent. However, the first proposal of this approach was defined upon a base process adapted from RUP. It focuses on modeling business processes and sub-processes, indicating how to derive the services from them. The methodological approach presented in this work provides a set of disciplines, activities, roles and work products, which are compatible with the three service modeling steps defined in SOMA: identification, specification and realization. This methodology is still under development and does not include the automatic support to derive services from business processes yet.

Method Development Kit for Service-Oriented Architecture (SOA-MDK)
SOA-MDK [7] is a method for service-oriented development based on an adaptation and enhancement of component based techniques. This approach proposes the application of model driven architecture principles withing the context of reference models with which to provide a methodology framework for developing systems based on SOA and Component Based Development. Barn *et al.* in [7] argue that many approaches focus only on the technical issues and software development practices required for SOA, however, to little efforts can be found in the earlier parts of the software lifecycle for SOA, especially in the analysis phase using model based approaches.

The SOA-MDK describes a functional viewpoint of the SOAs development, i.e. the capturing the functional requirements of the system from business process through to services specification. This approach defines a series of activities and tasks, such as develop process models, develop information models, factor process models, partition information model into services, allocate activities to services, specify service and generate WSDL specifications. The key element of the approach is to focus on models, however, the nature of the model-driven based of this approach remains unclear.

Service Centric System Engineering (SeCSE) [74]
SeCSE is a *European Integrated Project* that aims to create methods, tools and techniques for system integrators and service providers and to support the cost-effective development and use of dependable services and service-centric applications. The

main goals of this approach is to extend existing approaches to service and system specification to include: requirements modeling, quality of service, and dependability specifications, with which to be used for service discovery and binding mechanisms. Within this project free and open source instruments enabling the engineering of a service-centric system are provided. These instruments are: tools, methods and techniques supporting the cost-effective development and use of dynamic, customizable, adaptable and dependable services and service-centric applications. It describes an approach to service-oriented systems development lifecycle, however, does not comment on the use of the model-driven development paradigm.

SOA Repeatable Quality (RQ)
SOA RQ is a proprietary methodology by Sun Microsystems that is based on a RUP-like iterative and incremental process consisting of five phases: inception, elaboration, construction, transition, and conception. UML compliant artifacts are used for documenting various deliverables of these phases.

Service Oriented Architecture Framework (SOAF) [23]
SOAF consists of five main phases: information elicitation, service identification, service definition, service realization, and roadmap and planning. It is concurrently based on two types of modeling activities: "To-be" modeling, which is the top-down business oriented approach describing the required business processes, and "As-is" modeling, which is the bottom-up approach describing current business processes as they are shaped by the existing applications.

Thomas Erl's [22]
The service oriented analysis and design methodology is a step by step guide through the two main phases: analysis and design. The activities in the analysis phase take a top-down business view where service candidates are identified. These serve as input for the next phase, service oriented design, where the service candidates are specified in detail and later realized as Web services.

Steve Jones' Methodology for Service Architectures [36]
The scope of this top-down methodology consists of the first steps in a project necessary to ensure that true SOA properties are satisfied in the final delivery. It is technology agnostic and takes a top-down business view reaching up to the point of service candidate discovery (i.e. identification). The methodology adopts a broadly four-step process (What, Who, Why, and How), of which the first three are covered in preparation for the fourth step.

3.2.2. SOA-focused techniques

Although, to the best of my knowledge, methodologies covering a complete development process with requirements automated transformations do not exist, various techniques and methods for SOA-based software modeling and transformations especially at the PIM level can be found in the literature.

Delgado *et al.* [19] propose an approach which relates BPMN models with *Service Oriented Architecture Modeling Language* (SoaML) models, offering automated transformations between these two models implemented in a QVT (*Query View Transformation*) standard transformation language.

Ali *et al.* [3] present an approach for automatic transformations from the SoaML to a design model in the form of OSGi (*Open Services Gateway initiative*) Declarative Services Model.

In the service-oriented modeling context, another approach was presented by Cao *et al.* in [13]. In this proposal, authors define a service-oriented way to model the requirement and also a model refinement mechanism with a set of refinement rules. The refinement mechanism and rules can transform the requirement model to a set of loosely coupled services.

Jamshidi *et al.* [34] propose an innovative method called the *Automated Service Specification Method* (ASSM) which automatically generates the service model from service-oriented specifications. It consists of several tasks and steps which guide developers in performing the method. In order to accomplish the ASSM tasks, the *Automatic Service Identification Method* (ASIM) has to be performed on the CIM-level artifacts in order to execute further transformations from the business model to the service model.

Since business processes are an important elemet of the SOA, the automation of these business workflows has become an issue in software engineering. Zhang and Jiang in [76] describe a method for automated *Business Process Execution Language* (BPEL) generation from the business process specifications. This method consists of a number of well-defined mapping rules to describe a strategy for model transformation from business logic to BPEL.

Another approach of BPMN to BPEL transformations is presented in [21]. Within this approach the business process is expressed in an abstract model (Business Process Modeling Notation or BPMN) and according to transformation rules it is automatically mapped to an execution language (Business Process Execution Language or BPEL) that can be executed by a process engine. The authors of this work introduce the term business process oriented programming to refer to an evolutionary step in software engineering where programming power is given to the business analyst.

3.3. Agile methodologies

OpenUP/MDD

A more general-purpose software development process, still under development, is the OpenUP/MDD plug-in of the OpenUP methodology. OpenUP is a very simplified RUP version intended for small teams and which defines the minimal set of roles, tasks and artifacts. The OpenUP/MDD approach models the process conforming to the OMG's MDA [61]. It provides 6 role definitions, 57 different work products, and 90 standard task descriptions. Since it is consistent with the MDA, it focuses only on the transformations from the PIM to PSM level of the architecture and does not define a model-driven means of creating the PIM level requirements specifications. In this context, our proposal for the RUP extension and the OpenUP/MDD approach are complementary. The methodological approach presented in this work focuses on the CIM level transformations, generating a desired model at the PIM level. Specifying the CIM to PIM transformations reduces the system analysts' workload and responsibilities by including domain experts and stakeholders in the system modeling.

The Agile Unified Process (AUP)

AUP is a simplified version of the RUP which applies agile techniques to Agile Model Driven Development (AMMD), change management, and particularly focuses on Test Driven Development (TDD). It has a new Model discipline which encompasses the RUP's Business Modeling, Requirements, and Analysis & Design disciplines, and considers the model as the principal artifact of the requirements specifications. Change management activities have alo been moved from Configuration & Change Management to the Model discipline. However, this methodological approach does not provide well-specified activities and workflows for the model-driven development process, in an attempt to keep the process as agile as possible.

3.4. Requirements engineering in the health-care

Health-care is a domain characterized by highly complex processes and data. Its peculiaritis, such as complex stakeholders, the variability of health-care requirements, contexts and relations between them, dynamic requirements adaptable to changing over time care conditions and care relationships, or requirements from a distributed sources, describes in more detail Chapter 2. These characteristics make the requirements engineering in the health-care domain even more important than it usually is. Moreover, it becomes a very complex process. Possible errors at the RE stage can lead not only to badly designed systems, but also can significantly harm patientes. This section presents different particular techniques and more complex approaches to the

requirements engineering in the health-care domain.

Requirements elicitation

Health-care organizations demand a lot of effort to be put on developing health-care systems. Most of these software are critical and sensitive to mistakes because they involve people's lifes. If these mistakes come from the elicitation of requirements the harm that the system causes may be even greater. Several requirements elicitation techniques have been introduced in the literature having their background in the real projects development. Some on these elicitation techniques are the following:

- Document reading - this technique assumes that analysts read existing documents, job descriptions, tasks descriptions, quality assurance manuals, to get familiar with the domain vocabulary;
- Questionnaire - a questionnaire is a set of questions that different stakeholders have to respond;
- Interviews - requirements elicitation technique by conversation between software engineers, stakeholders, and future system users;
- JAD sections - is used more likely to solve conflicts than to gather knowledge about the domain; it assums a collaborative description of requirements to jointly produce the its final version;
- Protocol analysis and observation - protocol analysis consists on asking the stakeholders to talk about what they are doing while performing the task, while an observation of performed activities is a frequently used technique to confirm the understanding of the activity or process, described by the use of some other technique;
- Use cases and scenarios - wery frequently used during functional requirements elicitation; While use cases give an abstract view of the system to be created, scenarios are easier to be validated by stakeholders, who feel more comfortable with natural language scenarios descriptions;

Document reading, interviews, use cases and scenarios are the most frequently used. But neither of these techniques is sufficient to be applied individually. Many different elicitation techniques should be used to understand the requirements for projects in this complex health-care domain. In the context of requirements elicitation, Cysneiros in [15] comments on the importance of the understanding of the vocabulary used in the domain. To avoid misunderstandings caused by different concepts understanding, the use of some kind of vocabulary control, such as Language Extended Lexicon (LEL), is suggested. In the same work, Cysneiros argues for a special attention for non-functional requirements (NFR), which are not only important but also numerous. The most commonly described non-functional requirements of the domain are: safety, security, privacy, reliability, availability, and usability. Some of these NFRs can be

conflicting among them, so it is especially important to seach for such conflicts and solve them.

Requirements specification

Garde and Knaup in [28] present an approach to analyze and describe requirements for a given branch in the health-care domain. Their approach to requirements engineering combines a grounded theory approach with evolutionary prototyping based on the constant development and refinement of a generic domain model. They present the use of this approach on the example of chemotherapy planning in paediatric oncology. It is a part of the health-care domain, where chemotherapy planning tasks are very complex and time consuming, thus error at the requirements engineering stage must be avoided.

Grounded theory supports the inductive development of theories which are grounded in data, but also very complex and thus difficult to explain the theory phenomenon. A theory in this approach is developed by systematically recording and analysing data related to the phenomenon. This process is called *constant comparison*. The relevant for a theory data is gathered from different sources of evidence in order to develop a valid theory. That data is then compare to the previously collected data. If new aspects are uncovered during analysis in the particular field, the researcher stops the comparison and adds the new aspect to the initial data set. The process stops when new aspects of the studied phenomenon are not expected to appear. At this point it is worth noticing, that the theory created in this way is flexible for possible modifications if new apsects to the researched theory appears.

The requirements engineering approach for health-care proposed by Garde and Knaup in [28] bases on the grounded theory. As chemotherapy is characterized by many professionals and inidividuals involved, each one having particular requirements, the software system they use should be flexible enough to be adapted to these particular needs. The approach consideres creation of a generic domain model for specific knowledge of the domain (in this case chemotherapy planning in paediatric oncology). The main goal of the use of grounded theory in this approach is to support the process of requirements engineering by providing basis for a comprehensive domain model creation. To address this issue, the UML model has been used with which to perform the continuous requirements analysis, the specification prototyping, and the grounded theory refinement.

McGee-Lennon in [53] describes features that should be taken into consideration while describing requirements in the health-care domain. These features are the following:

- participatory elicitation and negotiation;
- distributed elicitation and negotiation;

- iteration affording rounds of eliciting, balancing and validating requirements;
- identification of and engagement with appropriate stakeholders to elicit high quality requirements;
- prioritisation or weighting of requirements;
- retention and traceability of requirements over time;
- identification and categorisation of requirements conflict;
- resolution of requirements conflict;
- annotation of requirements to enable both negotiation and traceability;
- correlation with other processes and work practices such as care assessment;

3.5. Related work summary

This chapter joins few important areas significant for the methodological appraoch of the present thesis: requirements engineering techniques and their use in the model-driven context, methodologies and methods for service-oriented development, and finally essential requirements engineering techniques of the health-care domain.

Cabot and Yu [12] argue that it is necessary to extend *Model-Driven Engineering* (MDE) methods with improved requirements techniques. To gather the current knowledge on the last decade research of the RE techniques in the MDD context, a systematic literature review was performed. Systematic literature review is the type of investigation, which is considered as one of the most accurate techniques to review the current state of researches in an objective and repeatable way. Research in the last decade has shown increasing progress with regard to the precision and automatic support that can be applied to requirements engineering specifications. However, a complete solution which includes requirements models as part of MDD processes is still lacking. Greater benefits could be obtained by applying model-driven transformations at the requirements level [57]. In addition, currently few tool support is provided with which to manage requirements models and to make further use of them in an automatic manner.

As the OpenUP/MDRE approach introduced in this thesis focuses on creating SOAs for the health-care systems, an overview on the most common service-oriented approaches is provided. Finally, requirements engineering techniques frequently used in the health-care domain are discussed. However, these researches mainly focus on elicitating understandable and non-conflicting requirements rather than their modeling and further use in the model-driven context.

Chapter 4

OpenUP/MDRE methodology

The OpenUP/MDRE presented in this work is an extension of the OpenUP software development methodology. It extends the agile OpenUP approach by putting it in the model-driven development context, to provide a development process that covers the requirements engineering tasks. It also emphasizes the importance of an architecture in the software development process. While all projects from different domains have different process needs, the OpenUP/MDRE provides a means to adapt the methodology to those needs in architecture-driven process engineering.

It has been developed as a plug-in library for Eclipse Process Framework (EPF) [2]. OpenUp/MDRE includes new content elements, such as: artifacts, roles, tasks, and processes elements, i.e., activities and capability patterns, to guide software engineers who attempt to follow an MDD approach in their software projects.

This chapter is organized as follows. The methodology main ideas are presented in the Introduction section, describing the new process in the context of the structure of the MDA framework and the MDA principles. Following sections introduce the building blocks of the OpenUP extension to incorporate the RE techniques in the complete MDD process. Moreover, the *model-driven requirements* discipline which constitute the main extension of the OpenUP methodology, with which to treat the requirements models is described in details. Finally, the *environment* discipline from the classic RUP is presented, as it is introduced in the OpenUP/MDRE as a necessary means of supporting the methodology adaptation to various architecture-driven development processes.

4.1. Introduction

As described in the previous chapters model-driven guidelines bring different point of view in the approaches for software development. The MDA differs basically from the traditional model of development process. The methodological approach for software production developed in this research has as a goal applying the MDA guidelines at the very beginning of the development process. Differences between the classic MDA-based approach and the one proposed in this work shows the Figure 4.1. In the classic process (Figure 4.1.A) the MDA-based process covers the software development starting from the analysis. Introduced here methodology extends the use of MDA framework lifecycle to include the requirements explicitly in the development process taking advantage of the CIM-level models (Figure 4.1.B).

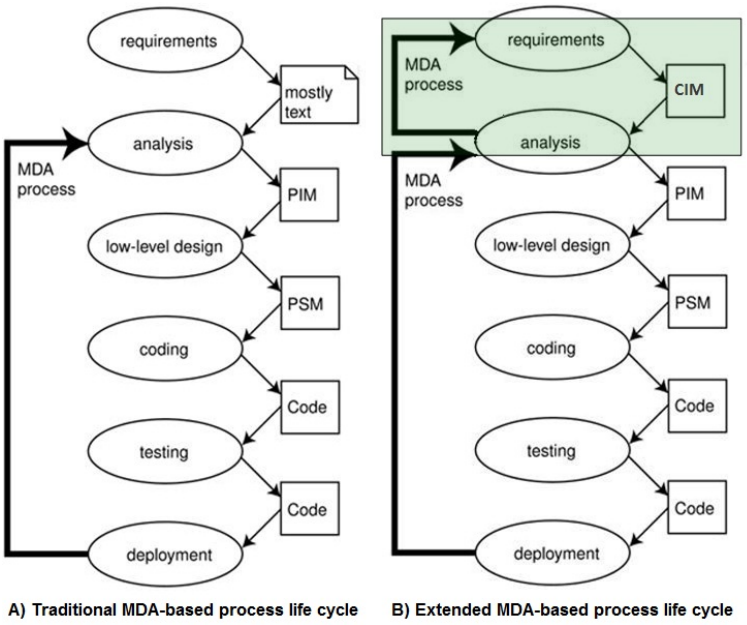

Figure 4.1. Traditional and extended MDA-based approaches comparison

Creating the CIM-level models allows an automatic transformation of the requirements with which to create the PIM-level specifications. To date the task of PIM creation was performed manually (Figure 2.4), thus very often causing inconsistencies in the PIM regarding to previously captured requirements. Including the CIM-level specification as a part of the complete MDA-based process is not the only novelty of the approach. In addition, the new methodology focuses on a very important element of the software development process which is the software architecture. On the basis of the identified for the project architecture, the CIM and the PIM representation is defined. Creating the most important artifacts of the requirements engineering applying the architecture-driven approach is an essential idea of this new methodological proposal. For this purpose a specific CIM-level model is created by qualified technicians. The

CIM later is transformed applying transformation rules to the PIM. Depending on the architecture chosen, one project can possess many corresponding PIMs, and not only one. All further steps of the MDA framework lifecycle remain without changes: the PSM is created on the base of the PIM, to finally generate the code for desired specific technological platform. The Figure 4.2 shows the schema of the OpenUP/MDRE methodological approach in terms of the MDA abstraction levels.

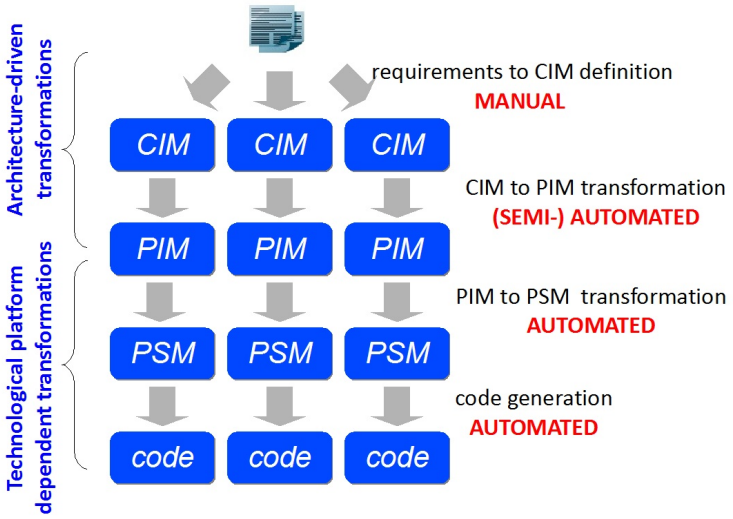

Figure 4.2. Extended schema of the MDA framework for OpenUP/MDRE

4.2. Methodology overview

The methodology describes a complete process of a software development as an instance of the OpenUP. It focuses on the requirements engineering as it is the software development critical process. In this approach, requirements are managed in an iterative process of their specification and further management. This process consists of activities workflow which initially lead to the user acceptance of collected requirements and stakeholder visions. Moreover, it includes activities with which to perform a generation of project specification in the model-driven context.
Within each iteration a specific activities belonging to different disciplines are executed. These iterations are organized into phases, offering a well-defined process workflow for each particular phase. The methodology was created as a EPF plug-in for the OpenUP. Some screenshots from this tool can be found in the Appendix C.
Two new disciplines, for requirements engineering activities and process configuration, are provided in comparison to the classic OpenUP. These disciplines add the agility to the development, making it adaptable to SOA-based development process and taking the advantage of the benefits which bring the model-driven development principles.

The Model-Driven Requirements discipline allows a smooth transition from the CIM-level model to the PIM-level specification. This transition is driven by the architecture identified, which conditionates the artifacts to be used and the transformation process to be followed. This architecture-oriented methodology has been defined to increase the efficiency of use of SOA-based systems in the health-care domain. However, the *Environment* discipline provided, makes it flexible enough to adapt this methodology to be used with other architectures and in different domains. The Environment discipline is a means of describing the configuration of artifacts which derive the model-driven development process to be followed. Commented here development process is completed with the OpenUP standard disciplines, such as: Architecture, Development, Test, Configuration and Change Management, and Project Management.

4.2.1. Disciplines

Figure 4.3 illustrates the extended version of the OpenUP hump chart. It shows the expected workload for each of the defined disciplines during iterations and phases that constitutes the project lifecycle. As depicted in Figure 4.3, the new discipline *Model-Driven Requirements* is the principal discipline, tasks of which are performed in the *Inception* phase. As a result of this discipline the product vision is created and the stakeholder requests are documented. The information collected within this discipline is crucial for the rest of activities of this stage from the Project Management and Architecture disciplines. Moreover, since we concentrate on model use in the MDD context, the workload in the Development discipline in the *Elaboration* phase decreases depending on the degree of automation of model transformations. A perfect MDD process decreases to minimum the necessity of coding in the *Development* tasks.

The *Architecture* discipline (marked by a star) is a discipline that is not always used during the OpenUP/MDRE lifecycle. When a necessity of a new architecture development arises, then the *Architecture* discipline is used to develop adequate architectural elements, models, or patterns. However, as the methodology is intended directly for the SOA, the *Architecture* discipline is optional and may be narrowed to refine the reference architecture provided. Architecture may depend on the domain of application or organization which reuse architectures developed in the previous projects. Health-care is one of those domains where systems architecture is mostly limited to SOA, as it is interoperable, reusable and flexible as the domain demands. For this reason the *Architecture* discipline in this work will not be used, however it is not excluded from the methodology standard disciplines.

Finally, as showed on the Figure 4.3, the Environment discipline is a concern from the *Inception* phase to the *Transition*. However, as the hump chart emphasizes, the workload within this discipline is especially important during the *Inception* phase, in which the software development process is configured and the environment for the project is established.

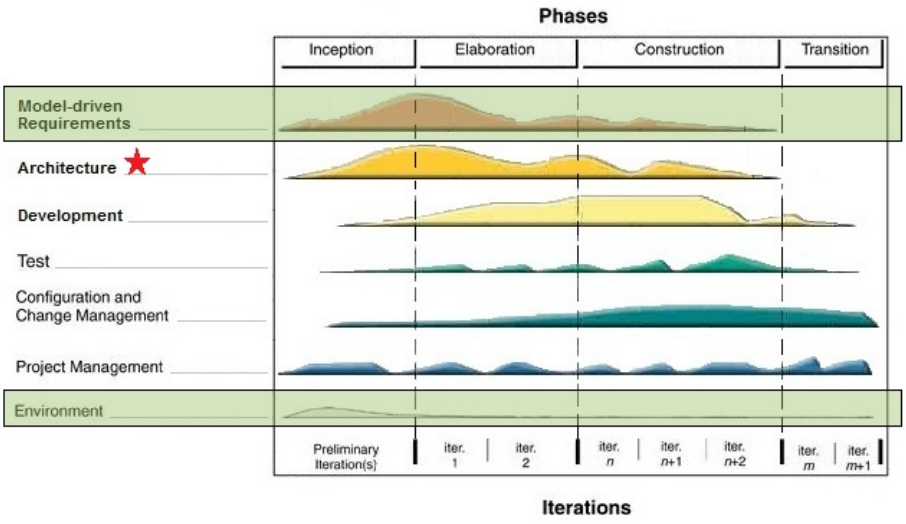

Figure 4.3. OpenUP/MDRE disciplines

4.2.2. OpenUP implementation structure

As mentioned in the previous chapters, OpenUP has been defined as a plug-in of the EPF [2]. It distinguish two essential elements of its structure: "Method Content" and "Processes". The former includes the definition of the necessary elements of the methodology (roles, artifacts, tasks, and relations between these elements), while the latter describes the process built from reusable capability patterns previously defined. The structure of the "Method Content" of the OpenUP/MDRE is the following.

The Method Content includes two main building blocks which are the Content Packages and Standard Categories. While the Content Packages describe in detail the elements of the methodology, the Standards Categories section gives an overview on the disciplines, work product kinds, role sets, and tools. As shown on the Figure 4.4, the content of the OpenUP methodology is classified into the following blocks:

- collaboration - method package which contains the foundation elements for OpenUP, reflecting the collaborative nature of the process;
- intent - method package which contains elements that deal with how to channel the intent of stakeholders to the rest of the development team, to ensure that validated builds with incremental capabilities reflect stakeholder intents; the model-driven requirements content forms a part of the intent package;
- management - method package which contains elements that deal with management of the project, including project planning, iteration planning, day-to-day management of the work within the iteration, and iteration assessments;
- solution - method package which contains elements that describe all aspects of creating the architecture, designing, implementing, and testing the application, as well as preparing the environment configuration for the specific project needs;

- templates - content package which contains templates documents with which to facilitate the artifacts creation in the entire project;

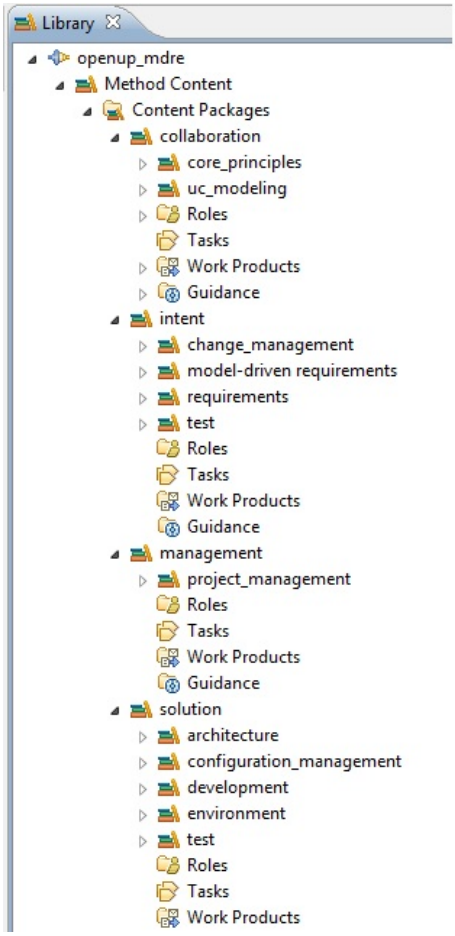

Figure 4.4. Content structure of the OpenUP/MDRE

4.3. Requirements Engineering in the OpenUP/MDRE

Requirements engineering is critical to successful software projects. Tasks performed at this stage of development process affect the entire development cycle. Many RE techniques have been introduced to increase the efficiency of this domain. However, approaches to automate the RE processes have not been defined. One of the ways in which to improve the development process is to focus on requirements engineering based on models in the model-driven context.

In order to address this issue, we propose the new discipline of OpenUP for Model Driven Requirements. The purpose of this discipline is to improve the effectiveness of the requirements discipline and eliminate all misunderstandings between customers, analysts and developers at both the essential requirements stage of software development and at all further development stages. In OpenUP, the original Requirements

discipline serves to establish the agreement with customers with regard to what the system should do and define boundaries of the system. It is a basis for planning and estimating the time and cost of the project. However, it should also provide a means for developers to better understand the requirements, it being like a bridge between the domain experts, stakeholders and the IT people.

The OpenUP extension for model-driven requirements engineering defines the new discipline within which to describe roles, work products, tasks and processes to capture, organize, model, transform and manage requirements. It includes: 2 additional roles, 7 new work products and a set of 15 tasks, from which 13 are new comparing to the classic OpenUP Requirements discipline. The following subsection describe in detail introduced here elements of the OpenUP/MDRE.

4.3.1. Roles

Model-Driven Requirements disciplines declares three roles in its *Method Content*: *Analyst*, *Model Analyst*, and *Transformation Specifier*. The Analyst is one of the principal roles originally defined in OpenUP, two remaining roles are new and constitute the main extension part. This section describes these roles that take part in the processes of model-driven requirements. The Table 4.1 gives an overview of roles, assigned to them tasks and work products for which they are responsible.

Analyst
The person in this role represents customer and end-user concerns by gathering input from stakeholders to understand the problem to be solved and by capturing and setting priorities for requirements. An Analyst needs the following knowledge, skills, and abilities:

- expertise in identifying and understanding problems and opportunities;
- ability to articulate the needs that are associated with the key problem to be solved or opportunity to be realized;
- ability to collaborate effectively with the extended team through collaborative working sessions, workshops, etc.;
- good communication skills, verbally and in writing;
- knowledge of the business and technology domains or the ability to quickly absorb and understand such information;
- knowledge of modeling languages and techniques;

An *Analyst* is responsible for the following artifacts: Glossary, Supporting Requirements Specification, Use Cases, and the Vision document. The main tasks that performs an *Analyst* are requirements elicitation tasks belonging to the inception phase and requirements analyzing tasks from the elaboration phase. However, the participation of an *Analyst* during all phases of development process is crucial is (s)he manages

changes in requirements and maintains traceability links performing the ongoing tasks. The tasks to which the *Analyst* role is assigned are the following: Capture Common Vocabulary, Elicit Stakeholders Requests, Find and Outline Requirements, Define Vision, Develop Requirements Models, Develop Supplementary Specifications, Manage Dependencies, and Manage Model Dependencies. As additional performer, an Analyst helps in architecture identification. A complete list of tasks and artifacts for which the *Analyst* is responsible is presented in Figure 4.5.

Figure 4.5. List of key artifacts and tasks to be performed by the Analyst

Model Analyst

During the *Model-Driven Requirements* discipline, the *Model Analyst* leads the activities of system modeling and transformation execution. The person in this role participates in and coordinates a number of tasks related to: model generation, model traceability (model dependency management) and model validation. The main artifact for which the *Model Analyst* is responsible is the analysis model. Its type of content depends on the architecture identified, while the model must suit the architectural pattern considered. For this reason it is recommendable that the *Model Analyst* possesses the following knowledge, skills, and abilities:

- expertise in meta-modeling and modeling languages, patterns, and tools;
- expertise in model-driven processes engineering;
- knowledge of model transformation techniques;
- knowledge of technology domains and the ability to quickly absorb and understand such information;
- knowledge of software architectures;
- ability to collaborate effectively with the extended team through collaborative working sessions, workshops, etc.;
- ability to articulate inconsistencies in the requirements documents;

A complete list of tasks and artifacts for which the *Model Analyst* is responsible is presented in Figure 4.6. This role also collaborates with the *Analyst* to accomplish a number of requirements traceability tasks, such as: *Manage Dependencies* and *Manage Model Dependencies*.

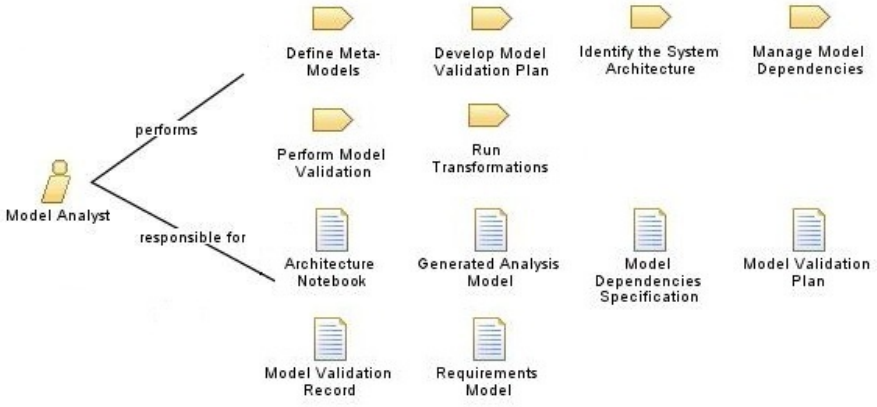

Figure 4.6. List of key artifacts and tasks to be performed by the Model Analyst

Transformation Specifier

This role is responsible for specifying the details of transformation rules to transform particular elements of the source model into corresponding elements in the target model. It is a good practice to establish such rules in the meta-models level, which also simplifies the traceability and model dependency management. These transformation rules are defined in the Transformation Rules Catalog (TRC) artifact. The *Transformation Specifier* is responsible for the artifacts related to the definition of model-driven transformations, what shows the Figure 4.7. A *Transformation Specifier* needs the following knowledge, skills, and abilities:

- expertise in meta-modeling and modeling languages, as also transformation languages, standards, and tools;
- expertise in model transformation techniques and implementation;
- knowledge of software architectures;
- ability to collaborate effectively with the extended team through collaborative working sessions, workshops, etc.;

Collaborating roles

A person in the role of an *architect* is responsible for the main architecture artifact, which also plays significant role in the process of requirements transformation. An *Architect* collaborates with *Model Analyst*, *Analyst*, and *Transformation Specifier* to accomplish the architecture identification for the project. S(he) also controls the Architecture Notebook collaborative creation while being responsible for this artifact.

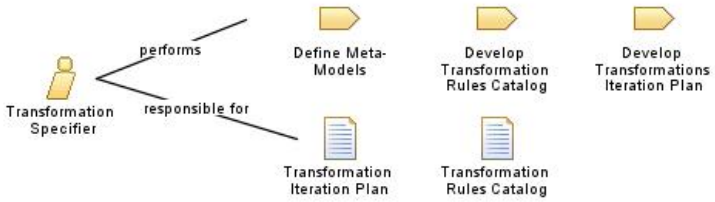

Figure 4.7. List of key artifacts and tasks to be performed by the Transformation Specifier

4.3.2. Artifacts

This section briefly describes the artifacts of the newly introduced Model-Driven Requirements discipline. We can distinguish the artifacts defined by the original requirements discipline (such as: Vision, Glossary, Supporting Requirements Specification, Use Cases, Work Items List) as also new artifacts with which to establish the necessary elements of a model-driven development process. The summary of these new work products can be found in Table 4.2.

Transformation Rules Catalog (TRC)
On the basis of the elements of the source and target models identified, the transformation rules are specified. This artifact should consist of a precise description of rules, elements mappings, refinements, which also provides the basis for the requirements traceability and model dependencies management. It should also provide some examples of use to facilitate its use by the Models Analyst.

Transformation Iteration Plan (TIP)
CIM to PIM transformations are usually quite complex and frequently are based on generating some intermediate models at the CIM level. For this reason not a single transformation, but a sequence of them is necessary. This artifact is created to plan a logical order of transformation to be performed.

Requirements Model (RM)
This model is the CIM-level model of the approach. It is the initial artifact to which the transformations apply and generate PIMs. The type of its content depends on the architecture identified, while the model must suit the architectural pattern considered and the process chosen to be followed.

Generated Analysis Model (GAM)
This model is the PIM-level model of the approach. It is the most important work product of the discipline, it being a source for further transformations to generate

Table 4.1. Roles of Model-Driven Requirements discipline

Role: ***Analyst***

Tasks	**Work Products**	**Principal objectives**
• Capture Common Vocabulary • Define Vision • Develop Requirements Model • Develop Supplementary Specifications • Elicit Stakeholder Requests • Find and Outline Requirements • Manage Dependencies • Manage Model Dependencies	• Glossary • Supporting Requirements Specification • Use-Case Model • Vision	• to capture user requirements • to refine requirements • to establish common vocabulary to communicate with various stakeholders • to understands particular users needs • to look for inconsistencies in requirements • to specify a clear definition of the requirements model • to manage requirements traceability

Role: ***Model Analyst***

Tasks	**Work Products**	**Principal objectives**
• Define Analysis Model • Define Meta-Models • Define Requirements Model • Develop Model Validation Plan • Identify the System Architecture • Manage Model Dependencies • Perform Model Validation • Run Transformations	• Architecture Notebook • Generated Analysis Model • Model Dependencies Specification • Model Validation Plan • Model Validation Record • Requirements Model	• to manage the model-driven RE process • to define an architecture that suits the project (as reusable as possible) • to define transformation process artifacts • to manage model dependencies, including traceability links between requirements and analysis models • to take the responsibility of models correctness • to perform model transformations

Role: ***Transformation Specifier***

Tasks	**Work Products**	**Principal objectives**
• Define Meta-Models • Develop Transformation Rules Catalog • Develop Transformation Iteration Plan	• Transformation Iteration Plan • Transformation Rules Catalog	• to define clear transformation rules • to collaborate with *Model Analyst* with regard to meta-model definitions • to use standard transformation languages (if possible) • to define transformations of high automation level

PSMs. The type of its content depends on the architecture identified, while the model must suit the architectural pattern considered.

Model Dependencies Specification (MDS)
The main purpose of this artifact is to specify dependencies between different types of models. Such document facilitates the requirements traceability throughout the entire development process, helping to maintain the specification consistent even when requirements specification changes occur.

Model Validation Plan (MVP)
Together with the TIP artifact, the model validation is planned in order to detect all inconsistencies in the specification of model transformations and failures in their execution.

Model Validation Record (MVR)
This artifact is to store the result of the Validate the Requirements Model task. It may be an interesting process management element, with which to discover the inaccuracies in the model transformations.

An important artifact which is developed in this discipline is the *Architecture Notebook* (initiated in the *Architecture* discipline) in which to store the information about the architecture and the models and meta-models that this architecture implies.

Table 4.2. Work products of Model-Driven Requirements discipline

Work product	**Responsible role**	**Modified by**
Requirements Model	Model Analyst	Analyst
Generated Analysis Model	Model Analyst	
Transformation Rules Catalog	Transformation Specifier	
Transformation Iteration Plan	Transformation Specifier	
Model Dependencies Specification	Model Analyst	Transformation Specifier
Model Validation Plan	Model Analyst	
Model Validation Record	Model Analyst	

4.3.3. Process - activities workflow

A set of new activities has been defined and a reference workflow has been designed for the new discipline. Some activities of this workflow, such as: Analyze the Problem,

Understand Stakeholders Needs, or Manage Changing Requirements, belong to the Requirements discipline of the classic RUP. Moreover, new activities that describe the new architecture-oriented model-driven approach were defined. This section presents a detailed overview of each of the activities from the Model-Driven Requirements discipline, including the description their internal tasks. Figure 4.8 demonstrates the Model-Driven Requirements discipline workflow represented through tailored version of a UML activity diagram.

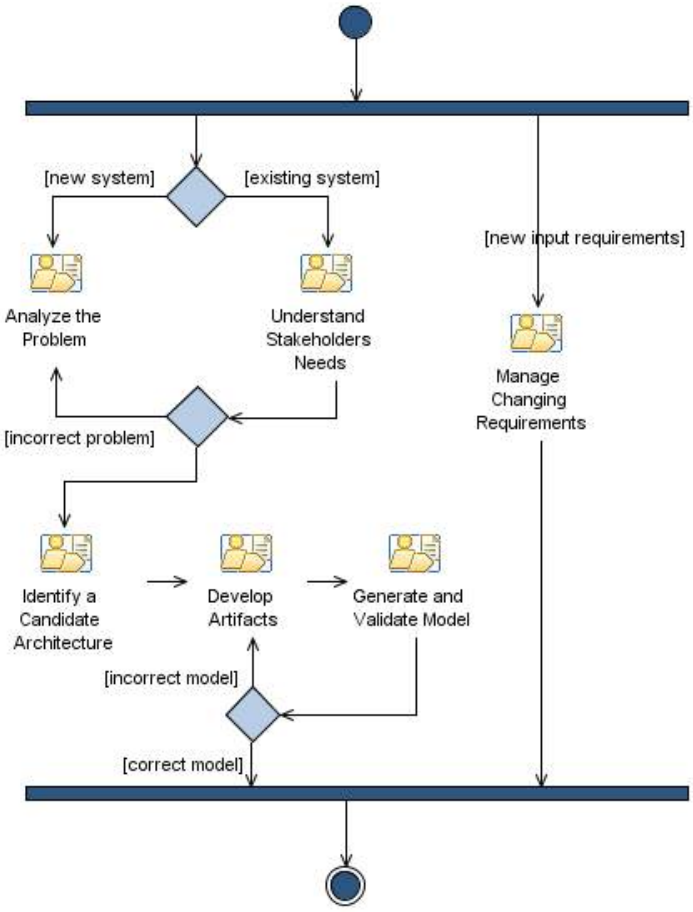

Figure 4.8. Model-Driven Requirements discipline workflow

Analyze the Problem

The purpose of this activity is to solve a new problem or to analyze a problem for which a solution already exists but addresses incorrect problem. Within this activity an agreement has to be made on a statement of the problem we are trying to solve, the stakeholders have to be identified and the boundaries and constraints of the system have to be clearly defined. To ensure the problem to be solved, the stakeholders have to be carefully identified for the better understanding of the need of each particular

stakeholder. The features which represent the high-level user or customer view of the system are captured in the Vision document. To minimize the potential of possible misunderstandings between the Analyst and the other different domains background stakeholders, a common vocabulary has to be established and maintained in the Glossary. To address this issues, the following tasks have been defined.

- Define Vision
- Capture Common Vocabulary

This activity belongs to the Inception phase of the project and is one of the principal activities that help the Project Manager in estimating project costs, time schedule and risks.

Understand Stakeholders Needs

The purpose of this activity is to capture all requests made on the project by different stakeholders, as well as specifying all the supplementary requirements that does not fit the functional specifications such as use cases. The stakeholder requests are usually captured during interviews, requirements elicitation workshops, change requests, statement of work, problem statement, laws, regulations, etc. Developers must understand the general vision on the requirements on which client (final user) agreed with the organization's representatives. After going through the stakeholder requests the Vision document should be reviewed as the Analyst acquires some new knowledge and some of the new stakeholder requests may have some impact those previously defined. To address this issues, the following tasks have been defined.

- Elicit Stakeholder Requests
- Develop Supplementary Specifications

This activity, similarly to the Analyze the Problem, belongs to the Inception phase. The artifacts that result from the performed tasks are the principal input for the system modeling in the further process.

Identify a Candidate Architecture

This is the main activity with which to establish the architecture for the project and the model-driven development process which this architecture implies. It is performed in the early *Elaboration* phase and is essential activity for the software development process in that it determine which artifacts need to be developed and the MDD process to be followed. Within this activity, the main architectural elements are identified and the type of model at the PIM-level is determined, these being the output of model-driven transformations.

If a project of a certain type is to be developed for the first time within an organization, this activity launches the Architecture discipline workflow with which to develop the new architecture description. If the identified architecture description already exists, the modeling artifacts that are implied by this architecture are defined. These

artifacts are: requirements model and a meta-model to which it conforms, as also the analysis meta-model defining the appropriate analysis model type. The information about the meta-models for further modeling and transformation purposes is stored in the *Architecture Notebook*. To address this issues, the following tasks have been defined.

- Identify the System Architecture
- Define Meta-Models

Develop Artifacts

As mentioned before, the architecture identified for the project conditionates the artifact types that will be used in the model-driven development process. These artifacts are models from the CIM- and PIM-level. The meta-models describing the types of these models at this stage is included in the Architecture Notebook document on the basis of which to develop transformation rules between these meta-models. The mappings between particular elements from the source model and particular elements of the target model are defined in the Transformation Rules Catalog. If these transformations can be automated, this document includes their the ready to use implementation in chosen transformations specification language (e.g. QVT, ATL). Depending on the complexity of source and target model, the transformations may also be less or more complex. Sometimes, several transformation steps may constitute the complete transformation process. For this reason, Transformation Iteration Plan is created with which to order the execution of transformation rules. In addition, on the basis of documents gathered by the Analyst during the requirements elicitation process, the requirements model is created conforming to the meta-model desired. This model is the source model for further model transformations. Within this activity the plan of model validation is optionally produced if such validation is planned. To address this issues, the following tasks have been defined.

- Develop Transformation Rules Catalog
- Develop Transformation Iteration Plan
- Develop Requirements Model
- Develop Model Validation Plan

Generate and Validate Model

The main purpose of this activity is the generate the principal artifact of development process which is the analysis model. On the basis of developed artifacts in the previous tasks, the transformations are performed. This transformations may be manual, or automated, depending on the level of their complexity. The transformation execution may be supported by appropriate tools which allow the execution of desired transformations. After creating the PIM-level model, its validation can be performed according to the model validation plan. To address this issues, the following tasks have been defined.

- Run Transformations
- Perform Model Validation

Manage Changing Requirements
This activity in the classic RUP is to ensure that the changes to the requirements are managed in an effective and efficient manner. To address this issue the management of the requirements traceability is necessary to evaluate the impact of each requirement change on the rest of the requirements. Within this activity the relationships between requirements are explicitly defined. Moreover, as the model-driven aspect of this approach implies the use of models to represent requirements, the traceability between these models has to be also maintained. Therefore Manage Changing Requirements is an activity which incorporates certain improvements with regard to the original one taken from the classic RUP, adding the explicit traceability links definition between models and particular model elements. To address this issues, the following tasks have been defined.

- Manage Dependencies
- Manage Model Dependencies

The summary of this discipline, presenting roles, performed by them tasks with associated to those tasks input and output artifacts, is shown in the Appendix B.

4.4. Methodology configuration

As mentioned in the introduction section of this chapter, the OpenUP/MDRE extension offers means to adapt the process to specific needs of a project or an organization. In the context of the architecture-oriented model-driven approach, the process to be followed during the software development can be adapted to the architecture identified. To address this issue, OpenUP/MDRE introduces the Environment discipline with which to prepare the software development process tailoring, necessary roles and artifacts selection, as well as the software and hardware environment configuration for the purposes of the project. It is a simplified version of the RUP's Environment discipline providing roles, artifacts and activities which are described next.

4.4.1. Roles

This section presents roles which participate in the Environment discipline of the OpenUP/MDRE approach. After a short characteristic of each role, associated artifacts and tasks are shown on the appropriate figure.

Process Engineer
The Process Engineer is the main role of this discipline. A person in this role is respon-

sible for the software development process itself. This includes configuring the process before project start-up and continuously improving the process during the development effort.

The most challenging task of the Process Engineer is to deploy a methodology in an organization or for a particular project. This includes configuring and customizing the methodology, and also involves adopting the current organization processes and standards. The most important artifact is the Development Case, which specifies the tailored process for the individual project. A *Development Case* describes how the project will apply the process within each of disciplines provided. For each process discipline, (s)he decides which artifacts to use and how to use them. A Development Case should be brief and refer to the process configuration for details. A complete list of tasks and artifacts for which the *Process Engineer* is responsible is presented in Figure 4.9.

Figure 4.9. List of key artifacts and tasks to be performed by the Process Engineer

Infrastructure Specialist

A person in this role is responsible for providing the infrastructure for the configured process. It includes software installations and hardware configurations (work stations, servers, networks) needed for the project development and testing. Small teams of developers may not have a separate person with exclusive dedication to perform aforementioned activities. It is more important in medium size and large industrial projects. A complete list of tasks and artifacts for which the *Infrastructure Specialist* is responsible is presented in Figure 4.10.

Figure 4.10. Artifacts and tasks associated to the Infrastructure Specialist role

4.4.2. Artifacts

This section describes the artifacts of the Environment discipline. The origin of these artifacts is the Environment discipline of the RUP [41] methodology.

Development Case
This is the principal artifact of the discipline used to describe the customizations made to the original development process for a specific project. It describes elements from different disciplines to define the Development Process tailored for the particular project. This artifact is initiated early in the inception phase and is later iteratively refined taking advantage from the lessons learned. The Development Case includes phases with associated milestones, artifacts to be used (how? who? when? level of detail?), activities to be performed and some description of the iteration plan.

Development Process
It is a configuration of the underlying methodology that meets the specific needs on the project. In the case of the OpenUP/MDRE methodology, development process mainly depends on the architecture selected for the project. The Development Process describes the process that a project must follow to produce the desired software as a result. It includes the definition of artifacts, responsible roles, and tasks to be performed. In the OpenUP/MDRE, establishing an artifact type means establishing model type as models are the primary artifacts of this model-driven approach. Requirements are represented as models, and the workflow of the development process from the rough vision document, through the requirements representation as models, analysis models, to the design models and finally code, depends on the established type of these models.

Project-Specific Guidelines
This artifacts is to describe guidelines on how to perform a particular task or activity in the context of the project. These guidelines may also include the use of standards and good practices, which very frequently are reused rather than created from the scratch for a particular project. In the case of the OpenUP/MRE methodology, such guidelines may concern the good practices of requirements model creation or the manner

of performing the model transformations, which can be executed automatically by the use of a specified tool, or can be performed manually by the appropriate specialist.

Project-Specific Templates
Templates are useful to create uniform documents by different people participating in the project. It is a good practice to provide a template for each artifact of the development process. In this way it will be easier to prepare and to understand by others project team members. Some methodologies prepare a base-template for an artifact, which is later tailored for the particular needs of the project.

User Support Material
An artifact that stands for the ease of the tailored process application and the use of hardware and software. It provides instructions for project participants on how to use the process, or tools.

Development Infrastructure
Includes the hardware and software on which the tools to run. Many project reuse the infrastructure of an organization rather than create an individual infrastructure for a specific project. Also ***Tools*** with which to perform the specific tasks of project modeling, development, testing, have to be specified.

4.4.3. Process workflow

The *Environment* discipline explicitly declares the use of the development configuration tasks at the beginning of the project and also at the beginning of each iteration. The workflow for this discipline contains three activities, which are the following:

Prepare Environment for Project
This activity is executed only during the Inception phase. It ensures that the appropriate process and tools have been chosen for a particular project. It includes tailoring the development process, developing artifacts for project-specific guidelines and templates, selecting and acquiring required tools, as well as preparing the development infrastructure. To address this issues, the following tasks have been defined.

- Tailor the Development Process for the Project
- Develop Development Case
- Prepare Guidelines for the Project
- Prepare Templates for the Project
- Select and Acquire Tools
- Launch Development Process

Prepare Environment for an Iteration
As the process as iterative, after each iterations the lessons learned should serve to refine the Development Case leading to the final refinement of the development environment. Moreover, in each iteration the responsible for the process person has to assure that the project-specific guidelines and templates meet the iteration needs. To address this issues, the Develop Development Case task has been defined.

Support Environment during an Iteration
Supporting the development environment is an activity that is performed until the project ends. It is to ensure that an appropriate environment is in place and the makes the developers job more efficient and effective. This activity might contain installing required software and verifying that the hardware installations work correctly. To address this issues, the following tasks have been defined.

- Set Up Tools
- Develop Support Materials
- Verify the Configuration and Installation
- Support Development

For the Evironment discipline workflow, that include described above activities, please refer to Figure 2.13 from Section 2.5.3.

The summary of this discipline, presenting roles, performed by them tasks with associated to those tasks input and output artifacts, is shown on the Figure B.4 in the Appendix B.

4.5. OpenUP/MDRE lifecycle

This section shows how the new activities of the newly introduced disciplines affect the entire process in particular phases of the project development. As tasks belonging to the *Model-Driven Requirements* and the *Environment* disciplines are performed mainly in the *Inception* and *Elaboration* phase, thus this section focuses on the workflows of these two phases.

4.5.1. Inception phase

This phase is the initial phase of the project where performed to achieve common agreements among the stakeholders on the project objectives. To goals of this phase are the following:

- to establish the project's scope and boundary conditions, including operational vision, acceptance and refusal criteria;

- to understand the critical functionalities of the system to be created;
- to identify the architecture to be applied in the project;
- to estimate the overall cost and schedule for the project;
- to identify potential risks;
- to configure the supporting environment, including process and tools, for the project;

To address this issues, the workflow from the Figure 4.11 has been introduced.

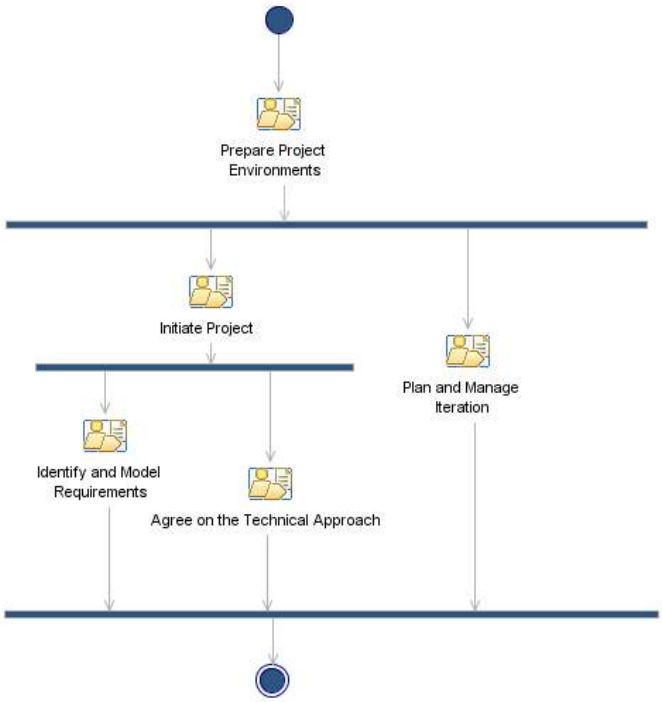

Figure 4.11. Activity diagram of the OpenUP/MDRE Inception phase

4.5.2. Elaboration phase

The *Elaboration* phase is characterized by strong focus on the architecture of the system. Most activities during a typical iteration in *Elaboration* phase happen in parallel. Essentially, the main objectives for *Elaboration* are related to better understanding the requirements, creating and establishing a baseline of the architecture for the system, and mitigating top-priority risks. As depicted on the activity diagram from the Figure 4.12, the requirements identification and modeling is performed simultaneously with the architecture development if such necessity exists.

Some of the main goals of this phase are the following:

- to establish a baselined architecture;
- to create the requirements model;
- to ensure that the architecture and requirements are consistent and stable;

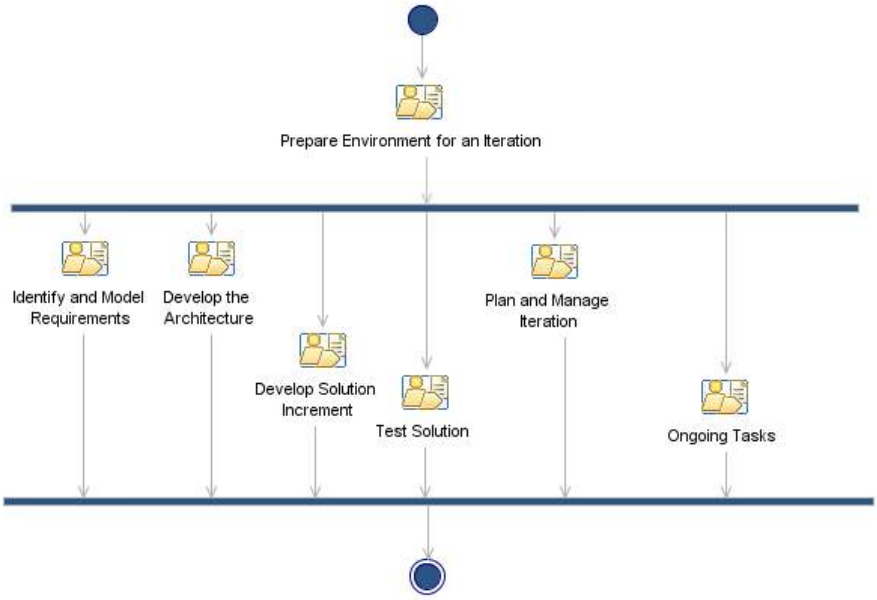

Figure 4.12. Activity diagram of the OpenUP/MDRE Elaboration phase

- to address all architecturally significant risks of the project;
- to establish the supporting environment;

Chapter 5

Case study

5.1. Case study context

This section provides a short description of the context of the case study with which to validate the methodological approach from the present thesis.

The case study chosen is based on my professional experience from the Indra Sistemas company from Valencia, Spain, where the MPOWER project [1] was developed. MPOWER is a user driven research and development project to create a middleware platform supporting rapid development and deployment of services for cognitive disabled and elderly. The platform definition process includes end-user requirements, design, platform development, development of proof-of-concept applications and end-user trials.

This project was financed by the European Union as a part of the European Union FP6 innovations research programme where a consortium of eight entities was led by norwegian independent research foundation SINTEF. The consortium also included: one of the world most important ICT companies (Ericsson Nicola Tesla), one large multinational company with employees in 17 countries (Indra Sistemas S.A.), one research center (Austrian Research Center), one university (University of Cyprus), and a medium size company (Tb-Solutions). Two entities experienced in similar projects provided real environments for the project testing and evaluation (Norwegian Center for Dementia Research, Norway and Medical College from Jagiellonian University, Poland). Indra Sistemas contributed to the project in many different fields taking advantage of the experiences gained in similar investigation projects. Indra was the leader of works related to design and implementation of the middleware for the integration of the smart house and sensors. Moreover, Indra had a significant contribution in defining the project architecture, business processes modeling and their configuration on the

integration environment. Also the tool-chain and proposed model-driven methodology evaluation was made by Indra's developers while identifying, designing and implementing the smart house and sensors related services.

The purpose of the MPOWER project was: firstly to define a development process for middleware services creation and secondly to implement using specified methodology a platform of reusable and interoperable services with which to facilitate the development of health-care applications. To address these issues the specified methodology used SOMA phases supported by some model-driven software development (MDSD) techniques. In particular, the methodology applied was based on the MDA framework and its guidelines which mainly concern the structure of software specifications in terms of models and their transformations. Transformations were applied in generating the platform-specific model and code generation. However, most of the work done on the services specification was done manually by developers.

The SOA was chosen as a base architecture for the project, as it is highly scalable, allows high reusability of the software components, as also is easily adaptable to new or changing user requirements. Having a set of services implemented and deployed on an environment supporting SOA, allows rapid and easy development process of health-care applications where the software personalization degree regarding to the user needs is very high.

Having prepared a set of middleware services, two proof-of-concept applications were designed. The end-users scenarios previously captured were classified into two groups: those containing social and information services, and those containing the use of sensors and medical devices in the context of a SMART HOME residence for cognitive disabled and elderly. Thus two proof-of-concept applications were designed and implemented.

The present case study demonstrates the development process of the actor management service, which is a base functionality in both of areas of interests of the MPOWER platform users. Defined for the purposes of the project process included the entire software development lifecycle: from the requirements elicitation to the applications implementation and testing. However, I will focus only on the requirements engineering activities, as RE is the main concern of the new OpenUP/MDRE methodological approach for the health-care systems.

5.2. Case study description (*Actor Management*)

This case study describes two different development approaches applied to a particular case from the user requirements captured within the MPOWER project. First commented development process example is an approach applied by developers within the MPOWER project. The second one, describes the OpenUP/MDRE approach proposed as a realization of the objective of the present thesis.

This case study discusses only the part of the development process which focuses on requirements engineering tasks with the final result of producing the analysis model which is the principal artifact on the PIM-level regarding to the MDA framework. Further development stages will not be commented as the principal concern of this OpenUP extension are the RE activities. However, it is noteworthy that the approach that was introduced to the MPOWER project follows the use of MDD processes in analysis to design models automated transformation as well as in the code generation.

The main objective of this case study is to validate the OpenUP/MDRE approach regarding its applicability and feasibility in a development process of health-care systems. To address this issue, differences between the original MPOWER approach and the OpenUP/MDRE are identified.

In both approaches, the system specification is developed on the basis of user requirements, which in the health-care are usually captured as user scenarios. These scenarios are given in a particular context which in the case of the health-care domain should describe actors participating in the particular scenario, their medical background and computer experience.
Scenarios of the present Actor Management case study, taken from the MPOWER requirements specification documents, are described next.

Actors description
Annie (68) and her husband Joe has been an active couple both in sports management and with following up their family. Annie has also been active in the board of the local Red Cross and is still active participating in specific activities.
Actors medical background and computer experience
From a medical point of view Annie will perform best if she can keep doing her normal daily activities. Annie does not have any software skills, but Joe often use the web and writes e-mail.

Requirements given as user scenarios
Scenario 1: Lately Annie has experienced problems with managing her everyday living due to early dementia, and she finds this very frustrating and the rest of her family does not feel comfortable with the situation. She forgets to take her medication, appointments, get anxious when is alone. For Annie and her family it is vital to create an environment where Annie is able to cope with her daily activities and where the family does not need to be afraid of her.
Scenario 2: The family is introduced to the different service providers that are involved in providing health and social services to Annie. The family starts using a secure, role-based and shared information space service that is provided by a local service provider. The shared information space makes all stakeholders able to share relevant information with Annie. An individual plan is created for Annie. The most important

activity that is added this week is Annie's medication plan. This plan was atomically added after the nurse had delivered the new medication. Every morning and evening Annie gets a reminder on her mobile phone.

The following sections describe how these scenarios are analyzed and processed applying both aforementioned approaches.

5.3. Applying the MPOWER approach to the case study

The development process in the MPOWER project, similarly to SOMA approach, has been divided into three subsequent phases: requirements specification, service specification, and service implementation. The workflow of activities within this process, emphasizing particular development phases, is shown in Figure 5.1 which will be described later in this section.

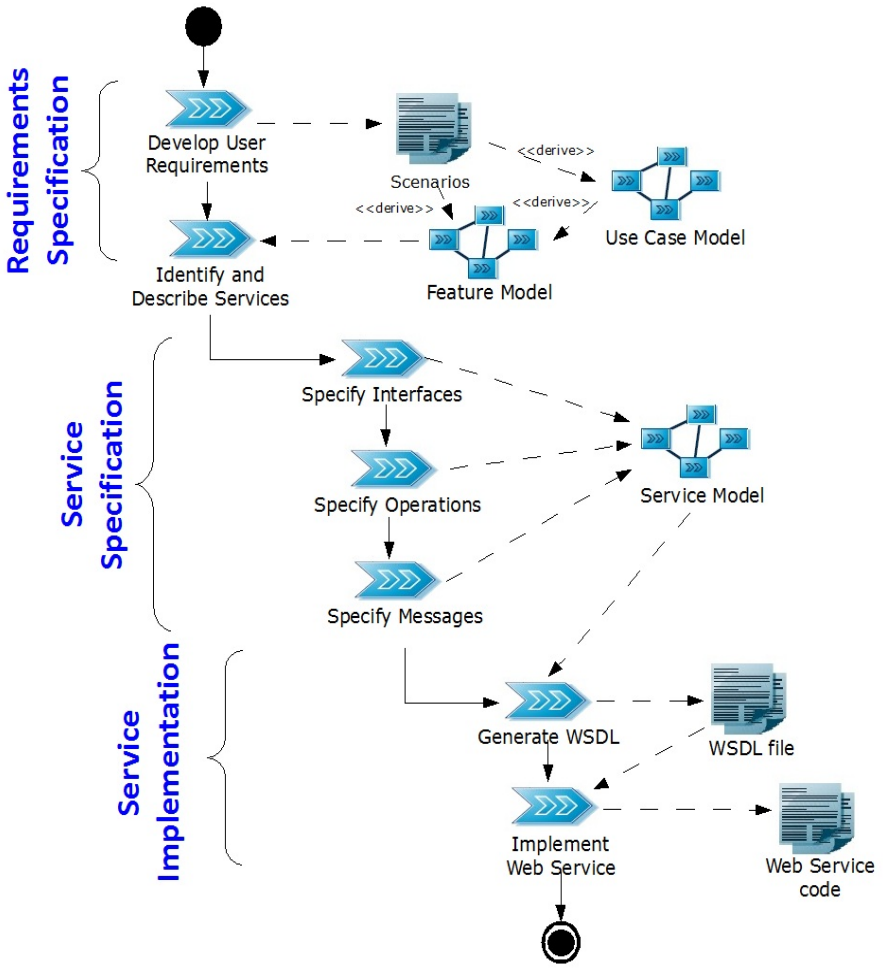

Figure 5.1. MPOWER services development approach.

Develop User Requirements

As mentioned in the previous chapters, requirements in the health-care domain are expressed mainly as textual scenarios description. Also in the case of the MPOWER project, health-care and home-care domain experts and the MPOWER proof-of-concept applications users were comfortable with this technique of requirements description. Thus, in the first phase of MPOWER development process, a set of user scenarios has been collected in the requirements elicitation process. These scenarios included description of health-care processes, actors identification, context of the occurrence of each scenario.

Identify and Describe Services

On the basis of aforementioned captured scenarios, analysts identified primary actors and use cases of the system and gathered this information in a use case model using UML standard notation. One of identified functionalities is stakeholders management. Figure 5.2.A shows an extract of the use case model created. This model served to identify features (see Figure 5.2.B, which are requirements descriptions called also higher-level expressions. A feature can be treated as a initial specification of a service to be provided by the system that directly fulfills a user need. This kind of specification was chosen to facilitate the process of services identification and their further specification. Because it is easy to discuss these features in natural language, to document and communicate them, they add an important context to the requirements engineering. Moreover. features may serve not only to specify functional requirements, but also non-functional constraints such as: risks, security limitations, reliability, etc.

Once the features are specified the process description assumes that services specification phase can be started. Identified services are traced from the features model. A kind of a manual mapping between the features and service was performed. Features were classified and grouped thematically. Finally, feature from the same group identified a service to be created. An example of such feature-service mapping is shown on the Figure 5.2.C.

Specify Interfaces, Specify Operations, Specify Messages

Having identified services to cover a specific user requirement, these services have to be specified. For this purpose, for each service, such elements as: interfaces, operations, and messages, have to be defined. This was realized by performing the following tasks: specify interfaces, specify operations, and specify messages, which resulted in creating the service model. The service model creation was done manually, taking care of the traceability links between features defined and services identified. An example of a service specified in this manner is shown in Figure 5.2.D.

Generate WSDL and ***Implement Web Service***

The MPOWER approach defines these two tasks as it describes an end-to-end process of services implementation. At the PSM-level, MPOWER takes advantage of the model-driven transformations applied to the analysis model previously created. However, this issue, as well as the services code generation, stay outside the scope of this case study.

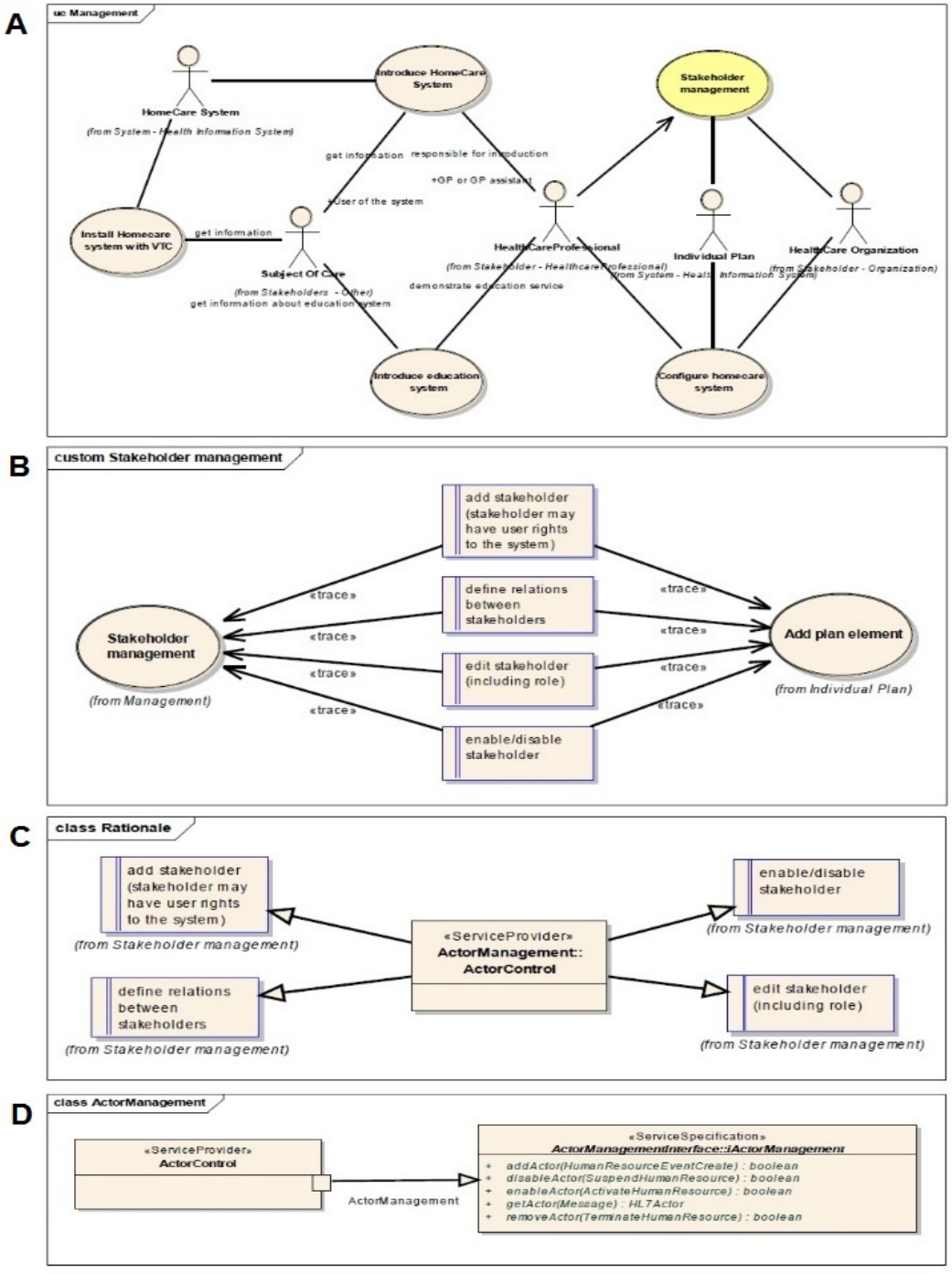

Figure 5.2. MPOWER service specification.

5.4. Applying the OpenUP/MDRE approach to the case study

In this section, the application of the OpenUP/MDRE methodology in the MPOWER project is discussed. As the OpenUP/MDRE methodology focuses on the architectural aspect of the system and the requirements engineering phase resulting in an analysis model, I will focus in this case study on these two disciplines of the methodology.

The *model-driven requirements* discipline provides a description of activities associated to them tasks with which to elicit, analyze and understand user requirements. It is the requirements understanding and refinement process that results in providing the process with an principal input in the form of textual specification and identified use cases.

In this case study, I assume that the requirements scenarios defined in the Section 5.2 are correctly captured and documented in the *Inception* phase of the project. Also use case model, similar to that from Figure 5.2.A, is created at this stage. These two artifacts constitute the input to the model-driven process that is presented in Figure 5.3. This figure shows a simplified schema of the designed process, presenting the main tasks, as well as their input and output artifacts. Tasks presented in this section cover the system modeling and model transformations activities which are performed in the *Elaboration* phase of the project.

Identify the System Architecture

This task is one of the most important tasks of the approach. Depending on its results, further efforts of developers can be decreased or increased. The former case occurs, when an architecture is identified that was previously used within an organization. In such situation the strategic elements (such as: meta-models, patterns, etc.) of the architecture are already defined and this task can be omitted. On the other hand, the developers' workload increases, when a specific architecture will be implemented for the first time within the development process of an organization. In this case, "Identify the System Architecture" task launches adequate procedure from the *Architecture* discipline with which to develop the architecture model.

In this case study of software development for health-care domain, the SOA architecture was a requirement for the middleware platform project discussed in previous sections. For this reason, the identification of a candidate architecture was limited to gather the experience of creating SOAs on the basis of the IBM SOA reference architecture [5]. To see some details about the SOA reference architecture please refer to the section 2.4.

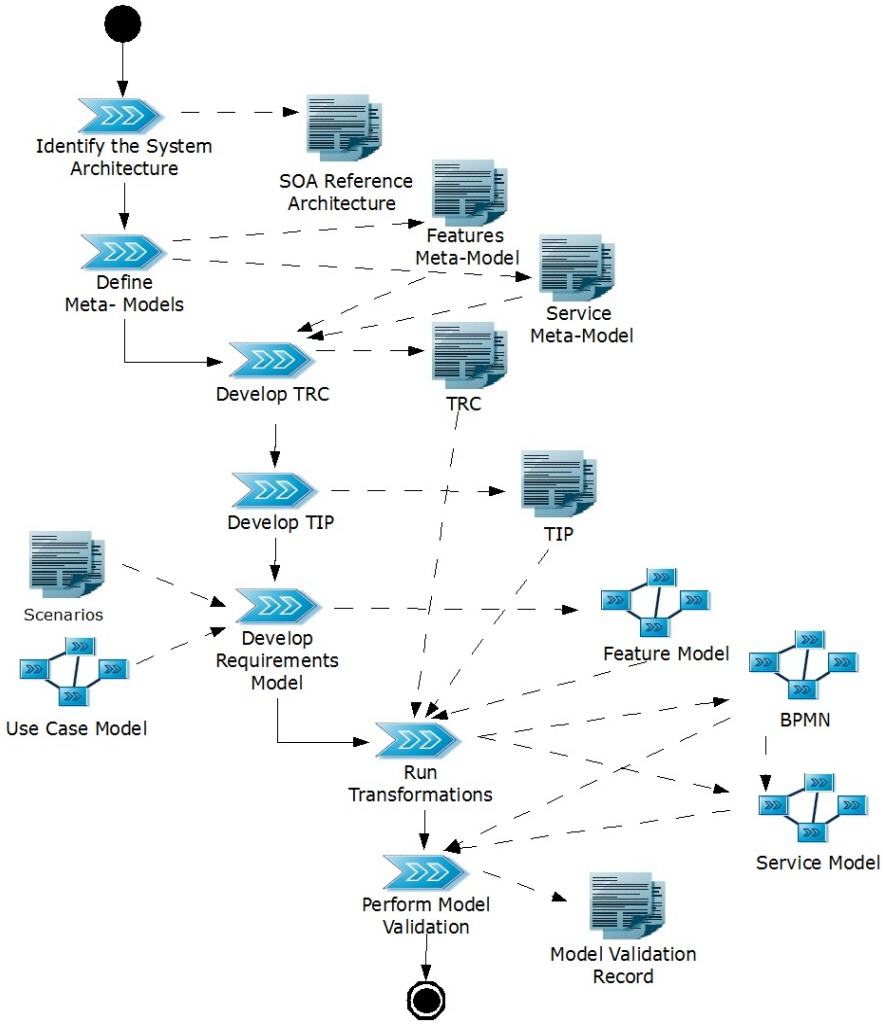

Figure 5.3. OpenUP/MDRE development process simplified workflow.

Define Meta-models

The person in the role of the Model Analyst defines the meta-models, conforming to which further requirements (CIM-level) and analysis (PIM-level) models will be created. In the case of the health-care domain, where requirements are described in natural language as user scenarios, the easiest way to describe requirements is the feature model. Its concepts describe services to be provided by the system. Feature meta-model is shown in Figure 5.4. It consists of service features described by a set of properties, such as: name, description, type, which are related between them using one of three refinement types: decomposition, specialization, and 'implement by' refinement. Also a number of constraints can be associated with each service feature.

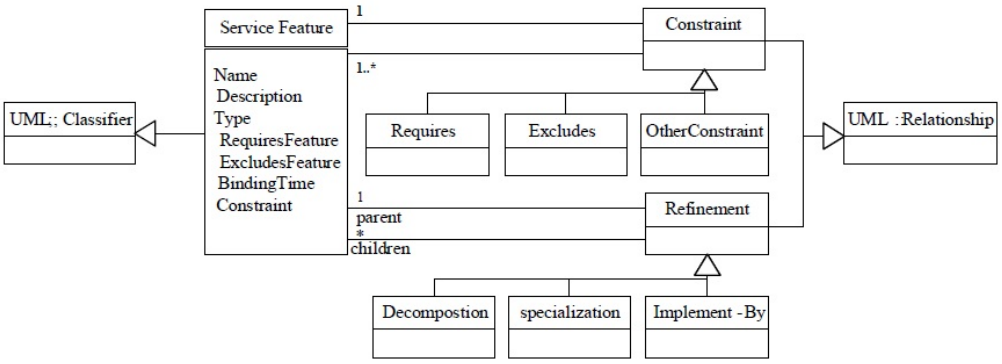

Figure 5.4. Feature meta-model.

In the case of the SOA architecture chosen for the project, the SOA architecture implies 'business process'-oriented and service-oriented models, to create the highly flexible and configurable software processes. As SOA is an architecture made of layers, it bases not only on the concept of a service, but also on an orchestration of services defined in the business process layer.
In this case, a meta-model for the analysis model would be the business process and the service model, conforming to the meta-models from Figures 5.5 and 5.6 respectively.

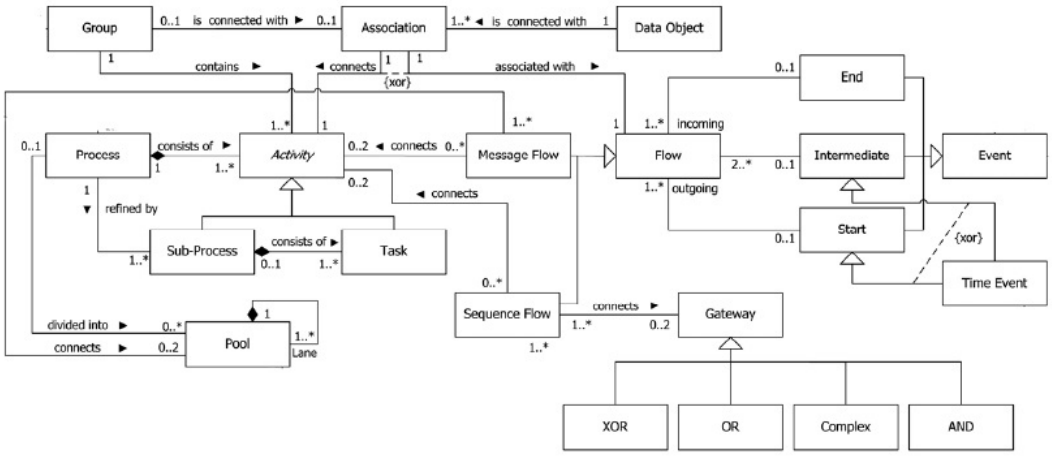

Figure 5.5. Business Process Modeling Notation meta-model.

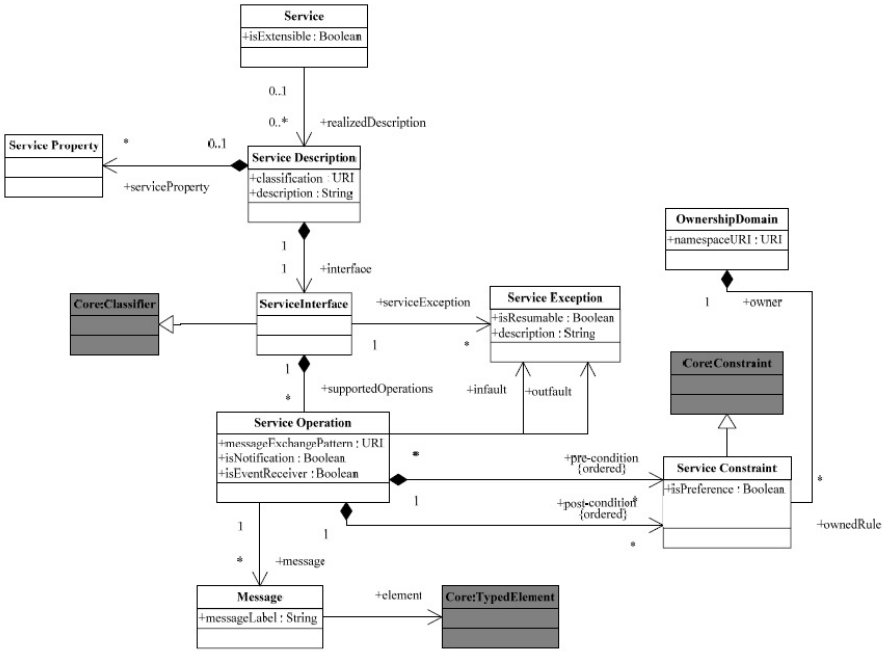

Figure 5.6. Service meta-model.

Develop TRC and ***Develop TIP***

The next step of this development process is creation of an artifact called *Transformation Rules Catalog* (TRC). In the case of choosing a feature model as a source model and a business process model and a service model as target models, the person in the transformation specifier role prepares a set of rules presented in this section. These rules, which are basically mappings between elements of both meta-models, can be executed automatically by the use of different kind of tools, but mainly they require to be done manually.

To transform the elements of the feature model to a business process model, transformation rules in the form of graphical elements mapping were defined. These rules are shown in Figure 5.7. For each element of the feature model, corresponding notation in both feature and BPMN models is presented.

Second work product to be obtained from the transformation process defined is the service model. The challenge here is in transforming BPMN models to a service model constructs. The approach that was chosen for this type of transformation is described by Azevedo *et al.* in [6]. This approach follows the SOMA phases of service development: identification, specification, and realization. During the identification phase, a candidate services are identified. These candidate services can be of two types: candidate data service and candidate business service. The former represents a service that realizes typical CRUD (Create, Retrieve, Update and Delete) operations

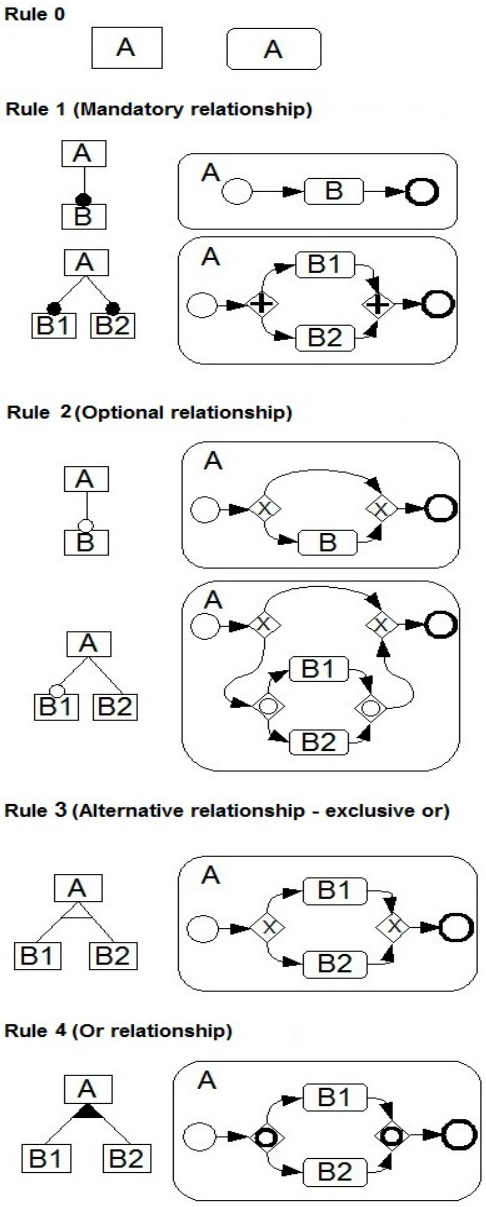

Figure 5.7. Mapping of feature model elements to BPMN.

on data objects that are stored in supporting data bases. The latter is a service that performs some business rules not related to CRUD operations.

The process of the services identification starts with selecting a set of activities from the business process models specification (BPMN). A process activity is selected if it is either automatic (performed entirely by a system with no manual interference) or partially supported by systems or automatable (manually executed, but expected to be supported by a system). Activities that are not being considered for automation are not selected, since it makes no sense to develop services for them.

Secondly, to the activities selected, several different rules (heuristics) can be applied to identify further service. Some of these rules are the following:

- A service must be identified from a business rule.
- A service must be identified from a business requirement.
- A service must be identified from a series of sequential activities.
- Operations must be identified from the interaction between two processes: one service to pass the information to the other process, and another service to receive the message.

For more detailed description of these heuristics please refer to [6].

At this point, it is worth to be mentioned that various transformations can be used to obtain the target model. More or less complex, more or less automated transformations depend on the *transformation specifier*. For instance, Elvesaeter *et al.* in [20] present a well-defined mapping between the BPMN and SoaML specification language, that could be adapted in the transformation rules catalog of OpenUP/MDRE and service-oriented health-care systems.

After defining all the rules to be used in the model transformation process, transformations order has to be set if the transformation process is not atomic and not straight forward. In the case of example presented in this chapter, two types of transformations have to be performed in order to obtain the final PIM-level analysis models. First, the feature model is transformed into the business process model, and after that from the business process model described in BPMN, services are identified and specified. This order is the primary topic of the TIP document. However, TIP may also include some information about the transformations supporting tools and the required models format (e.g. XMI [60]) with which to interchange models between tools for various transformations.

Develop Requirements Model

Requirements model creation process is a manual process of feature model creation on the basis of: i) user scenarios captured and ii) use cases defined in the use case model. This model is created in collaboration by the *Analyst* and the *Model Analyst*. An example of a simple model for the StakeholderManagement use case is shown in Figure 5.8.A.

Run Transformations

This task gets as its input the feature model created previously, and after performing a series of manual mappings and transformations, two output models are generated. From the CIM-level requirements model (Figure 5.8.A), first a business process model is created (Figure 5.8.B), and then after performing a sequence of predefined steps, the corresponding service model is obtained (Figure 5.8.C). Depending on the level of automation of the transformation rules, this task can be manual, semi-automated or automated. In the case of service model definition, services very often have to be

primarily identified and secondly specified manually by the analyst. However, this manual process can be given a set of precise guidelines, which make the process more trustworthy and faultless.

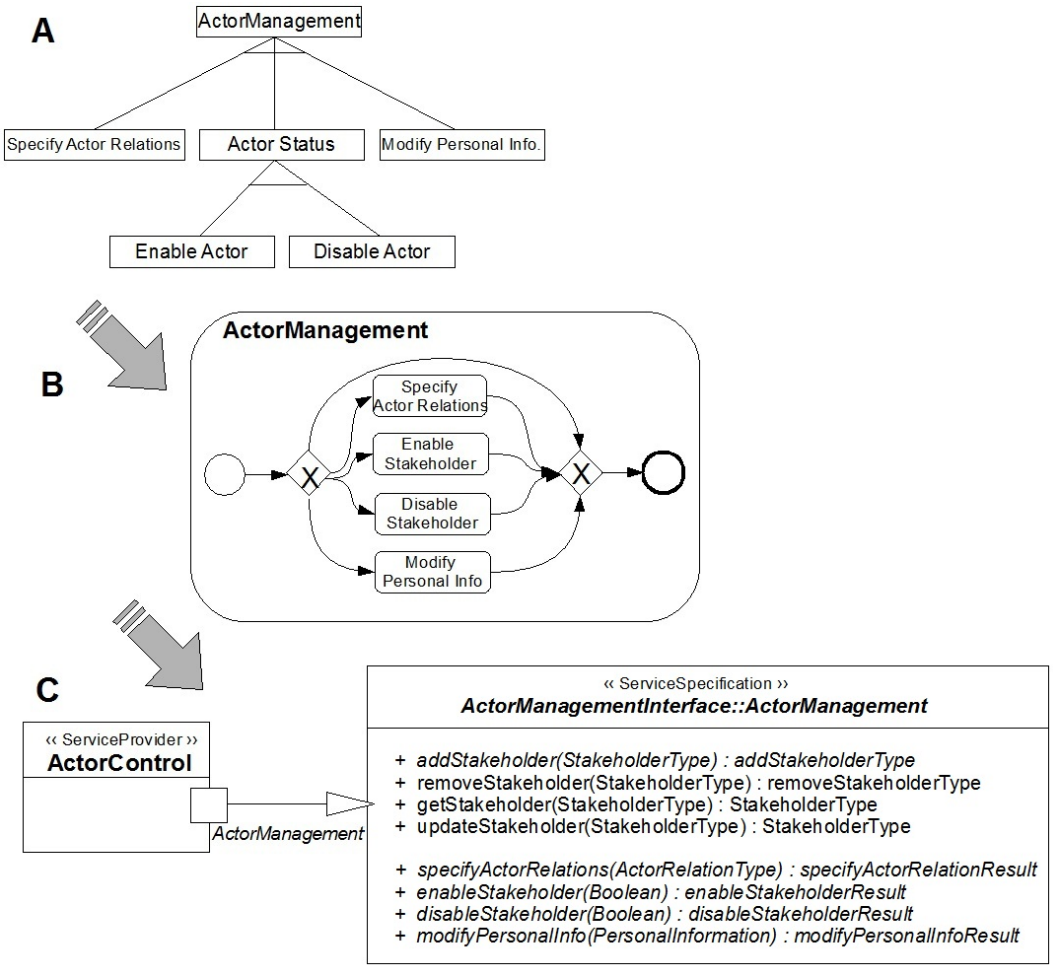

Figure 5.8. OpenUP/MDRE process applied to StakeholderManagement use case.

Perform Model Validation

This task can be performed in two ways: manually and automatically. It is an optional task in this process, however, a manual model validation is always recomended. Also verification that the new model keeps satisfying the user requirements should be performed by the *Analyst* and *Model Analyst*. In this case study the validation task will not be detailed while the service model validation against the health-care domain ontology standard HL7 [32] will be a part of the future improvements of the OpenUP/MDRE for health-care.

5.5. Case study conclusions

In this chapter, two processes of creating specifications for SOA are presented. The first one, was defined and performed within the MPOWER project. The second one, was the principal subject of this master thesis. This section comments on differences between these two development processes and also discusses some experiences in the lessons learned section.

5.5.1. Process comparison

Both processes described in this chapter were defined for the purpose of creation a middleware platform of health-care services. In both approaches the initial set of requirements are given as a set of user scenarios described in the natural language, which is closer to the health-care domain experts than any other requirements modeling language. Both software development approaches are based on the MDA framework lifecycle and employ models in further development stages as a base of concepts representation.

The main difference between these two approaches is the manner of creating models at the CIM and PIM level. The first approach neither provides any guidelines on how these models should be created nor defines any relations between these models. OpenUP/MDRE at this point is much more elaborated and controlled. It provides a well-defined process for requirements model creation by analysts at the CIM-level, and also defines further transformation steps to finally obtain the analysis model at the PIM-level. The process is supported by a well-documented meta-model-based structure definition for both requirements and analysis models.

Another important difference is that the service specification tasks in the MPOWER approach are by no means connected to the requirements specification products. Service candidates are identified and traceable to the user requirement documents, but these documents are of little use in further services specification. What means, that interfaces, services, but especially operations and their messages are developed at developers' own will. At this point, OpenUP/MDRE clearly defines the relation between requirements and analysis specifications.

Figure 5.9 illustrates a schema for both approaches, emphasizing the automation level of subsequent stages in the development process. As depicted on this figure, the business process design model (specified as BPEL) in the MPOWER process is created manually by the developers, whereas the service design model (specified as WSDL) is created by the use of an automated transformation. This may cause some inconsistencies between models, especially when one model uses elements of the other model at the implementation stage.

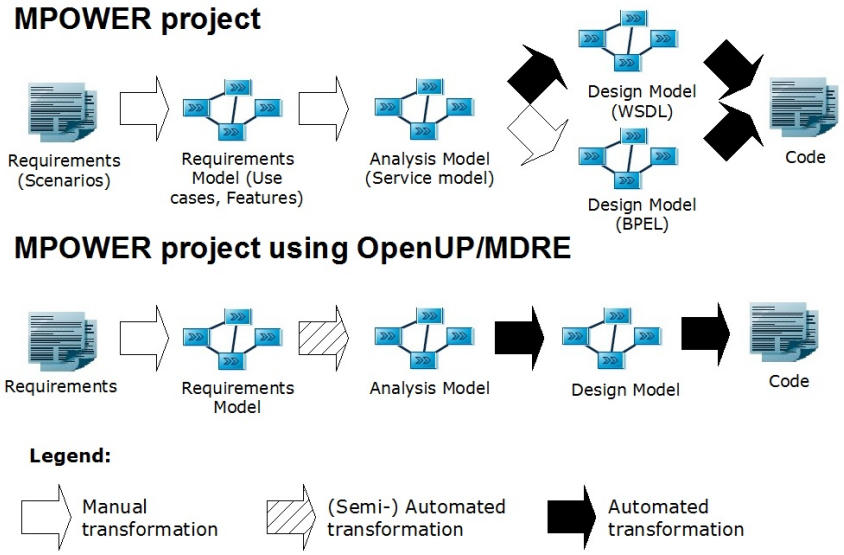

Figure 5.9. Case study processes automation. Comparison of approaches.

5.5.2. Lessons learned

This section presents lessons learned i.e. the knowledge gathered during the project realization. It was collected as one of the results of the project development, being a valuable help in introducing some improvements for more efficient realization of similar projects.

The main conclusion that can be drawn from realized MPOWER project and also performed here case study is that model-driven transformations can be applied no only at the PIM level, but also at the CIM level. Applying CIM to PIM transformations allows to control the analysis model consistency and requirements traceability to a larger degree than while creating the analysis model manually from the scratch. Moreover, the OpenUP/MDRE in relation to the MPOWER development process approach provides the following improvements:

- The traceability links between requirements and services are easier to maintain, while services identification and specification is done in controlled way, using identified rules.
- As the selected for the project architecture was SOA, MPOWER platform development was focused on implementing the two principal layers of this architecture: business process and service layers. However, MPOWER did not provide developers with a complete process of business processes modeling. That is why there have been discovered many inconsistencies between the implementation of these two layers, where services specification process was running in parallel to the business processes design (as BPEL). At this point, OpenUP/MDRE defines from the beginning of the development process two principal analysis artifacts to be obtained, which are developed maintaining the relation between them. A

top-down approach was proposed in this methodology, while only those services are specified which find application in one of the business processes previously defined.
    - Many services that have been implemented during the MPOWER realization remained unused.
    - Some services implemented across the organization were redundant (similar or the same), as some misclassification of the system functionalities occurred.
    - Some services were discovered useful in the proof-of-concept applications implementation stage, but have not been identified and implemented earlier.
- The MPOWER model-driven services development process was focused on code generation tools, rather than model transformation techniques.

During the MPOWER project realization a survey on the use of proposed model-driven process was performed. This study identifies factors that are important to developers while applying the MDD approach. The MPOWER developers after using described in this case study MPOWER development approach argue that the traceability feature in the model-driven approaches is crucial. The findings also suggest that perceived usefulness and ease of use are the most important factors for using a MDD development approach, therefore a well-defined process with guidelines and templates should be provided to developers.

Chapter 6

Conclusions

In this chapter the research held within the master thesis work is presented. In addition, further investigations on the OpenUP/MDRE methodology are identified. Finally, this chapter contains a summary of publications related to the present research.

Software systems are becoming more and more complex, and the success of their development no longer depends on individual effort and heroics. Successful software development can only be accomplished by using a well-defined software development process. Software engineers need guidelines to assist them in the software development process. These guidelines form a methodology that should help developers to facilitate the process of developing a complex software.

The *Requirements Engineering* process is recognized as being the most critical process in software development. Errors made during this process may have negative effects on subsequent development steps, and on the quality of the resulting software. However, many software development projects fail owing to a lack of well-defined methods and techniques describing the RE processes. For this reason, it is very important to have a methodology that clearly and precisely defines the RE phase.

A variety of requirements representations that are currently being used can be found in the literature. However, most approaches still use rather informal textual descriptions than models or other formal specifications.

Several approaches have been introduced in order to facilitate the software development process. One of these approaches is the Model-Driven Development. It is another approach that improves the development process of complex applications. It increases the productivity without augmenting the project management work load as also allows building applications of high quality faster and cheaper. It promotes the separation of concerns between the business specifications and their implementation. This separation is obtained through the use of models that allow the level of abstraction

to be elevated. However, the model-based foundation of this approach does not entail its use in the requirements discipline. It rather finds its application in the analysis and design phases, as well as in the implementation allowing code generation.
Menzies in his editorial [57] claims that models in MDD should be treated like a laboratory where analysts can experiment with different options (at the requirements and design level) of model design obtained automatically from requirements models. In the same work, Menzies talks over the advantages that should bring us experimenting with Model-Based Requirements Engineering (MBRE) which among others serve to comment and assess of the RE model, reach a resolution that better satisfies feuding stakeholders, ensure better decisions, etc. He also points some disadvantages of the use of requirements artifacts as part of the MDD processes. He indicates that it often takes too long to build the RE models so the process expenses increase and also that even if early lifecycle models exist, then no effective conclusions can be drawn from them since they are incomplete and full of contradictions and overlaps.

Many software development organizations producing highly complex software for equally complex domains, face the problem of performing requirements engineering tasks minimizing the number of failures in the requirements specification. One of such a complex domains is the health-care domain for which various methodological approaches have been introduced, but an ideal faultless process for requirements engineering has not been defined yet. As reported in [25] by the European Public Health Alliance, the recent problem of health-care domain is the limited accessibility of health-care services for the citizens. The solution is introducing information systems which will be highly interoperable and accessible. This requirements can be satisfied by information systems based on the Service-Oriented Architecture which gains its popularity in the health-care in the last decade. However, designing and implementing SOA-based solution are not trivial tasks. There exist many approaches to develop SOAs, but most of them focus only on services implementation, rather than providing adequate requirements traceability support or proposal for consistent modeling of business processes and services specifications.

The issues discussed here constitute the starting point for the principal contribution of this master thesis. This contribution is the OpenUP/MDRE approach: methodology for SOA-based application in the health-care domain which integrates requirements engineering techniques within a model-driven development process.

Within the research of this master thesis, the following tasks have been defined in order to reach the principal objective.

- *Task 1*: Analyze the use of requirements engineering techniques in the model-driven development processes.
- *Task 2*: Investigate on the current approaches of requirements engineering field in the health-care domain.

- *Task 3*: Define a generic methodology that builds upon the model-driven development principles, and facilitate the requirements engineering discipline.
- *Task 4*: Adapt proposed methodology for the purposes of the SOA-based health-care middleware.
- *Task 5*: Implement the methodology in a process engineering tool with which to provide the process definition easily accessible by developers.
- *Task 6*: Validate the approach, its feasibility and accuracy to the addressed domain;

With regard to Task 1, a systematic literature review (SLR) with which to analyze current use of the model-driven approaches which cover the requirements engineering activities, was performed. Within this SLR the following research questions were defined: "what requirements engineering techniques have been employed in model-driven development approaches and what is their actual level of automation?". 72 papers were chosen for the final review out of a set of 884 potentially related papers that have been searched. These papers were analyzed and classified regarding 12 criteria identified, such as: type of requirements used, type of requirements structure, transformations provided, transformations automation level, use of a standard language for transformation implementation, traceability provided, traceability automation, tool support provided, actual usage, evaluation method. After the review was performed, a series of conclusions has been drawn summarizing the current state of researches related to aforementioned topic. In addition, some research gaps have been identified.

With regard to Task 2, a comprehensive study about the current approaches of requirements engineering field in the health-care domain was performed. This study, that identified principal areas of interests of practitioners from the industry, confirmed the principal problems of the RE in health-care to be identical with those that come my personal professional experience in the domain. The conclusion drawn on the base of this study was a valuable factor in this master thesis motivation.

With regard to Task 3, a generic methodology that incorporates requirements engineering techniques in the context of model-driven development process was defined. The first proposal of the methodology was defined upon a base process adapted from the Rational Unified Process [41]. However, the agility which provides the OpenUP methodology was later found desirable, and the proposal base changed from RUP to OpenUP. OpenUP was found adequate methodology, providing extension mechanisms and architecture-oriented development process. In order to define this extended OpenUP methodology, the Unified Method Architecture (UMA) methods have been used.

With regard to Task 4, mechanisms for the methodology adaptation to specific needs of an organization or a project were defined. This mechanism is based on the environment discipline from the RUP which was adapted to be used in a simplified

form within the OpenUP extended methodology. The presence of the environment discipline adds some flexibility to the process which apart of being model-driven, is also architecture-oriented. This method was used to describe the discipline principal artifact (development case) for the SOA-based system from the health-care domain.

With regard to Task 5, the OpenUP/MDRE (OpenUP for Model-Driven Requirements Engineering) methodology was implemented as a plug-in of the OpenUP library from the Eclipse Process Framework [2]. In this way a configurable methodology description was prepared, which can be easily accessible as a web page for all project development members. It is also a helpful means for project managers and developers who attempt to follow an MDD approach in their software projects.

With regard to Task 6, defined methodology was validated by a case study taking as an example a real industrial project previously implemented. By comparing the new approach and the one used in the MPOWER project, I demonstrated the feasibility of the approach. However, further validation studies of this proposal are planned and are described in the next section.

6.1. Future work

The methodology definition presented in this master thesis is a first approximation of the model-driven development process for the health-care domain middleware systems. Nonetheless, there exist a necessity of its further improvements and evaluation. Some possible evolution direction for the OpenUP/MDRE methodology presents the following list.

- It would be desirable to provide the methodology with a support of a tool with which to easily create artifacts demanded for the model-driven development process (transformation rules catalog, transformation iteration plan, model validation plan, etc.). To address this issue, documents' templates and artifacts creation with wizards can be provided.
- Propose a health-care domain specific service models validation on the base of the HL7 [32] standard for health-care services.
- Propose a method for the requirements model (CIM-level) validation against an ontology of the health-care domain concepts.
- Validate the approach by measuring the effort involved in the methodology artifacts creation and the maintainability of requirements.
- Validate the approach by measuring the number of failures caused by errors in preparing the requirements specification in comparison to other similar size projects carried out with the use of classical methodologies.
- Collaborate with SINTEF in the investigation on the usability of the proposed methodology in real health-care projects in Norway. The validation tasks (more

case studies realization) may be also performed in cooperation with this research organization.
- Collaborate with Technical University of Munich in defining an artifact-driven approach for RE in MDD context.
- Investigate possible integration of the OpenUP/MDRE with the OpenUP/MDD which is a complementary approach to the present one, that covers PIM to PSM transformations.
- Extend this approximation for a development of software product lines in the health-care domain.

6.2. Related publications

During the development of the present master thesis, different contributions have been realized as publications. The following list gathers these publications which are placed in the order of their importance. For each publication a corresponding thesis part is indicated.

MODELS Conference 2010 (full paper)
Loniewski, G., Insfran, E., Abrahão, S.: A Systematic Review of the Use of Requirements Engineering Techniques in Model-Driven Development, Lecture Notes in Computer Science, In: Proceedings of the MODELS 2010: ACM/IEEE 13th International Conference on Model Driven Engineering Languages and Systems, Oslo, Norway, Lecture Notes in Computer Science, vol. 6395, pp. 213–227. Springer (2010)

This conference is the most relevant conference in the area of Model-Driven Software Development (MDSD). The MODELS '10 Conference acceptance rating: 21% (54 accepted papers out of 253).
Proceedings of this conference appear on the CiteSeerX[1] database [414/581] with the estimated impact of 0.01 based on Garfield's traditional impact factor.

This paper presents the systematic literature review, realized with the objective to gather current knowledge about the requirements engineering methods and techniques that are being used in the context of the model-driven development. This paper is directly related to the content of Section 3.1.

Method Engineering Conference 2011 (accepted 6-pages short paper)
Loniewski, G., Armesto, A., Insfran, E.: Incorporating the Model-Driven Techniques in the Requirements Engineering for Service-Oriented Development Process, In: Pro-

1. http://citeseerx.ist.psu.edu/stats/venues/

ceedings of the ME '11: International Conference on Method Engineering 2011, Paris, France

This conference is recognized as one of the principal conferences in the area of methods engineering.

This paper presents a software development methodology which incorporates the requirements engineering techniques for a service-oriented architectre in development of health-care systems. This paper proposes the RUP extension as the principal method. This paper is directly related to the content of Chapter 5.

EC-MDA 2008 Conference Workshop
Walderhaug, S., Mikalsen, M., Benc, I., Loniewski, G., Stav, E., Factors affecting developers' use of MDSD in the HealthCare Domain: Evaluation from the MPOWER Project, In: 3rd Workshop "From code centric to model centric software engineering: Practices, Implications and ROI" (C2M 2008) accompanying the ECMDA 2008 Conference Workshop (4th European Conference on Model Driven Architecture Foundations and Applications), Berlin, Germany (2008)

This paper presents experiences of the MPOWER developers from the use of MDD process in a real project of the health-care domain. It investigates which factors are important for developers to use MDSD in their work. This paper is directly related to the content of Section 5.5.2 and also Section 3.1.1.5.

ECMFA 2011 Conference (sent full paper)
Loniewski, G., Armesto, A., Insfran, E.: RUP Extension for Model-Driven Requirements Engineering, Sent to: 7th European Conference on Modelling Foundations and Applications, Birmingham, UK

This paper presents the idea of the RUP methodology extension for the model-driven requirements engineering techniques. This paper is directly related to the content of Chapter 5.

List of Figures

List of Tables

Abbreviations

ACM	ACM Digital Library
AUP	Agile Unified Process
BPM	Business Process Modeling
CBD	Component Based Development
CIM	Computation-Independent Model
CRM	Customer Relationship Management
CWM	Common Warehouse Metamodel
EPF	Eclipse Process Framework
EPHA	European Public Health Alliance
ERP	Enterprise Resource Planning
ESB	Enterprise Service Bus
GAM	Generated Analysis Model
IE	IEEE Xplore (digital library)
ISSI	Ingeniería del Software y Sistemas de Información
J2EE	Java 2 Platform Enterprise Edition
LEL	Language Extended Lexicon
MBRE	Model-Based Requirements Engineering
MDA	Model-Driven Architecture
MDD	Model-Driven Development
MDE	Model-Driven Engineering
MDS	Model Dependencies Specification
MDSD	Model-Driven Software Development
MODELS	Model Driven Engineering Languages and Systems (conference)
MOF	Meta Object Facility
MVP	Model Validation Plan

MVR	Model Validation Record
NFR	Non-Functional Requirements
OOAD	Object-Oriented Analysis and Design
OSGi	Open Services Gateway initiative
PIM	Platform-Independent Model
PSM	Plarform-Specific Model
QoS	Quality of Service
QVT	Query View Transformation
RE	Requirements Engineering (conference)
RE	Requirements Engineering
REJ	Requirements Engineering Journal
RM	Requirements Model
RMC	Rational Method Composer
RUP	Rational Unified Process
SD	Science Direct (digital library)
SL	SpringerLink (digital library)
SLR	Systematic Literature Review
SOA	Service-Oriented Architecture
SOAF	Service-Oriented Architecture Framework
SoaML	Service-Oriented Architecture Modeling Language
SOC	Service-Oriented Computing
SOMA	Service-Oriented Modeling and Architecture
SOUP	Service-Oriented Unified Process
SPE	Software Process Engineering
SPEM	Software Process Engineering Metamodel
TIP	Transformation Iteration Plan
TRC	Transformation Rules Catalog
UMA	Unified Method Architecture
WS	Web Service
WSDL	Web Services Definition Language
XMI	XML Metadata Interchange

Bibliography

[1] MPOWER project official web site, www.mpower-project.eu

[2] The Eclipse Foundation web site, The Process Framework (EPF) Project, http://www.eclipse.org/proposals/beacon/

[3] Ali, N., Nellipaiappan, R., Chandran, R., Babar, M.A.: Model driven support for the service oriented architecture modeling language. In: PESOS '10: Proceedings of the 2nd International Workshop on Principles of Engineering Service-Oriented Systems. pp. 8–14. ACM, New York, NY, USA (2010)

[4] Arsanjani, A., Ghosh, S., Allam, A., Abdollah, T., Gariapathy, S., Holley, K.: SOMA: a method for developing service-oriented solutions. IBM Syst. J. 47(3), 377–396 (2008), http://www.cs.jyu.fi/el/tjtse54_09/Artikkelit/ArsanjaniEtAlIBMSsJ.pdf

[5] Arsanjani, A., Zhang, L.J., Ellis, M., Allam, A., Channabasavaiah, K.: Design an SOA solution using a reference architecture (March 2007), http://www-128.ibm.com/developerworks/library/ar-archtemp/index.html

[6] Azevedo, L.G., Santoro, F., BaiĂŁo, F., Souza, J., Revoredo, K., Pereira, V., Herlain, I.: A method for service identification from business process models in a soa approach. In: Aalst, W., Mylopoulos, J., Sadeh, N.M., Shaw, M.J., Szyperski, C., Halpin, T., Krogstie, J., Nurcan, S., Proper, E., Schmidt, R., Soffer, P., Ukor, R. (eds.) Enterprise, Business-Process and Information Systems Modeling, Lecture Notes in Business Information Processing, vol. 29, pp. 99–112. Springer Berlin Heidelberg (2009)

[7] Barn, B., Dexter, H., Oussena, S., Sparks, D.: Soa-mdk: Towards a method development kit for service oriented system development. In: Magyar, G., Knapp, G., Wojtkowski, W., Wojtkowski, W.G., Zupancic, J. (eds.) Advances in Information Systems Development, pp. 191–201. Springer US (2008)

[8] Biffl, S., Mordinyi, R., Schatten, A.: A model-driven architecture approach using explicit stakeholder quality requirement models for building dependable information systems. In: WoSQ '07: Proceedings of the 5th International Workshop on Software Quality. p. 6. IEEE Computer Society, Washington, DC, USA (2007)

[9] Boulanger, J.L., Dao, V.Q.: Requirements engineering in a model-based methodology for embedded automotive software. pp. 263 –268 (july 2008)

[10] Brereton, P., Kitchenham, B.A., Budgen, D., Turner, M., Khalil, M.: Lessons from applying the systematic literature review process within the software engineering domain. J. Syst. Softw. 80(4), 571–583 (2007)

[11] Brown, A.W., Iyengar, S., Johnston, S.: A rational approach to model-driven development. IBM Syst. J. 45, 463–480 (July 2006)

[12] Cabot, J., Yu, E.: Improving requirements specifications in model-driven development processes, `http://jordicabot.com/papers/ChaMDE08Yu.pdf`

[13] Cao, X.X., Miao, H.K., Xu, Q.G.: Modeling and refining the service-oriented requirement. In: TASE '08: Proceedings of the 2008 2nd IFIP/IEEE International Symposium on Theoretical Aspects of Software Engineering. pp. 159–165. IEEE Computer Society, Washington, DC, USA (2008)

[14] Cleland-Huang, J., Hayes, J.H., Domel, J.M.: Model-based traceability. In: TEFSE '09: Proceedings of the 2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering. pp. 6–10. IEEE Computer Society, Washington, DC, USA (2009)

[15] Cysneiros, L.M.: Requirements engineering in the health care domain. In: RE '02: Proceedings of the 10th Anniversary IEEE Joint International Conference on Requirements Engineering. pp. 350–356. IEEE Computer Society, Washington, DC, USA (2002)

[16] Davis, A., Dieste, O., Hickey, A., Juristo, N., Moreno, A.M.: Effectiveness of requirements elicitation techniques: Empirical results derived from a systematic review. In: RE '06: Proceedings of the 14th IEEE International Requirements Engineering Conference. pp. 176–185. IEEE Computer Society, Washington, DC, USA (2006)

[17] Debnath, N., Leonardi, M., Mauco, M., Montejano, G., Riesco, D.: Improving model driven architecture with requirements models. In: ITNG 2008: Fifth International Conference on Information Technology: New Generations, 2008. pp. 21 –26 (april 2008)

[18] Delgado, A., Ruiz, F., de Guzmán, I.G.R., Piattin, M.: A Model-driven and Service-oriented framework for the business process improvement. Journal of Systems Integration vol. 1(3) (2010)

[19] Delgado, A., Ruiz, F., de Guzmán, I.G.R., Piattini, M.: MINERVA: Model drIveN and sErvice oRiented Framework for the Continuous Business Process improVement and relAted Tools. In: ICSOC/ServiceWave Workshops. pp. 456–466 (2009)

[20] Elvesaeter, B., Panfilenko, D., Jacobi, S., Hahn, C.: Aligning business and IT models

in service-oriented architectures using BPMN and SoaML. In: Proceedings of the First International Workshop on Model-Driven Interoperability. pp. 61–68. MDI '10, ACM, New York, NY, USA (2010)

[21] Emig, C., Weisser, J., Abeck, S.: Development of soa-based software systems - an evolutionary programming approach. Advanced International Conference on Telecommunications / Internet and Web Applications and Services, International Conference on 0, 182 (2006)

[22] Erl, T.: Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall PTR, Upper Saddle River, NJ, USA (2005)

[23] Erradi, A., Anand, S., Kulkarni, N.: Soaf: An architectural framework for service definition and realization. Services Computing, IEEE International Conference on 0, 151–158 (2006)

[24] Escalona, M.J., Gutiérrez, J.J., Rodríguez-Catalán, L., Guevara, A.: Model-driven in reverse: the practical experience of the aqua project. In: EATIS '09: Proceedings of the 2009 Euro American Conference on Telematics and Information Systems. pp. 1–6. ACM, New York, NY, USA (2009)

[25] European Public Health Alliance: EU CITIZENSHIP REPORT 2010 (October 2010), http://www.epha.org/

[26] Fenton, N.E., Pfleeger, S.L.: Software Metrics: a rigorous and practical approach. International Thompson computer, 2nd edn. (1996)

[27] Fliedl, G., Kop, C., Mayr, H.C., Salbrechter, A., Vohringer, J., Weber, G., Winkler, C.: Deriving static and dynamic concepts from software requirements using sophisticated tagging. Data & Knowledge Engineering 61(3), 433 – 448 (2007), advances on Natural Language Processing - NLDB 05

[28] Garde, S., Knaup, P.: Requirements engineering in health care: the example of chemotherapy planning in paediatric oncology. Requir. Eng. 11(4), 265–278 (2006)

[29] Gotel, O.C.Z., Finkelstein, C.W.: An analysis of the requirements traceability problem. Proceedings of the First International Conference on Requirements Engineering pp. 94–101 (April 1994)

[30] Haumer, P.: IBM Rational Method Composer: Part 1: key concepts (December 2005), http://www-128.ibm.com/developerworks/rational/library/jan06/haumer/

[31] Hinchey, M.G., Rash, J.L., Rouff, C.A., Gracanin, D.: Achieving dependability in sensor networks through automated requirements-based programming. Computer Communications 29(2), 246 – 256 (2006), dependable Wireless Sensor Networks

[32] HL7 International: HL7 Reference Information Model (2007), http://www.hl7.org/

[33] Insfran, E., Pastor, O., Wieringa, R.: Requirements engineering-based conceptual modeling. Requirements Engineering Journal 7, 61–72 (2002)

[34] Jamshidi, P., Khoshnevis, S., Teimourzadegan, R., Nikravesh, A., Shams, F.: Toward automatic transformation of enterprise business model to service model. In: PESOS '09: Proceedings of the 2009 ICSE Workshop on Principles of Engineering Service Oriented Systems. pp. 70–74. IEEE Computer Society, Washington, DC, USA (2009)

[35] Jones, M.R.: Computers can land people on mars, why can't they get them to work in a hospital? implementation of an electronic patient record system in a uk hospital. In: Methods of Information in Medicine. vol. 42, pp. 410–415 (2003)

[36] Jones, S., Morris, M.: A methodology for service architectures (Octobre 2005), `http://www.oasis-open.org/committees/download.php/15071/`

[37] Kherraf, S., Lefebvre, E., Suryn, W.: Transformation from cim to pim using patterns and archetypes. Software Engineering, 2008. ASWEC 2008. 19th Australian Conference on pp. 338 –346 (march 2008)

[38] Kitchenham, B.: Procedures for performing systematic reviews. Tech. rep., Keele University and NICTA (2004)

[39] Knethen, A.v.: A trace model for system requirements changes on embedded systems. In: Proceedings of the 4th International Workshop on Principles of Software Evolution. pp. 17–26. IWPSE '01, ACM, New York, NY, USA (2001), `http://doi.acm.org/10.1145/602461.602465`

[40] Koch, N., Zhang, G., Escalona, M.J.: Model transformations from requirements to web system design. In: ICWE '06: Proceedings of the 6th international conference on Web engineering. pp. 281–288. ACM, New York, NY, USA (2006)

[41] Kruchten, P.: The Rational Unified Process. Addison Wesley (1999)

[42] Laguna, M.A., Gonzalez-Baixauli, B.: Requirements variability models: meta-model based transformations. In: MIS '05: Proceedings of the 2005 symposia on Metainformatics. p. 9. ACM, New York, NY, USA (2005)

[43] Lapouchnian, A., Yu, Y., Liaskos, S., Mylopoulos, J.: Requirements-driven design of autonomic application software. In: CASCON '06: Proceedings of the 2006 conference of the Center for Advanced Studies on Collaborative research. p. 7. ACM, New York, NY, USA (2006)

[44] Letier, E., van Lamsweerde, A.: Deriving operational software specifications from system goals. In: SIGSOFT '02/FSE-10: Proceedings of the 10th ACM SIGSOFT symposium on Foundations of software engineering. pp. 119–128. ACM, New York, NY, USA (2002)

[45] Li, X., Liu, Z.: Prototyping system requirements model. Electronic Notes in Theoretical Computer Science 207, 17 – 32 (2008), proceedings of the 1st International Workshop on Harnessing Theories for Tool Support in Software (TTSS 2007)

[46] Liew, P., Kontogiannis, K., Tong, T.: A framework for business model driven development. In: Software Technology and Engineering Practice, 2004. STEP 2004. The 12th International Workshop on. pp. 8 pp. –56 (sept 2004)

[47] Loniewski, G., Insfrán, E., Abrahão, S.: A systematic review of the use of requirements engineering techniques in model-driven development. In: MoDELS (2). pp. 213–227 (2010)

[48] Machado, R., Fernandes, J., Monteiro, P., Rodrigues, H.: Transformation of uml models for service-oriented software architectures. In: ECBS '05: 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, 2005. pp. 173 – 182 (april 2005)

[49] Mairiza, D., Zowghi, D., Nurmuliani, N.: An investigation into the notion of non-functional requirements. In: SAC. pp. 311–317 (2010)

[50] Marschall, F., Schoenmakers, M.: Towards model-based requirements engineering for web-enabled b2b applications. In: ECBS'03: 10th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems. pp. 312–320. IEEE Computer Society, Huntsville, AL, EUA (April 2003)

[51] Mauco, M., Leonard, M., Riesco, D., Montejano, G., Debnath, N.: Formalising a derivation strategy for formal specifications from natural language requirements models. Proceedings of the Fifth IEEE International Symposium on Signal Processing and Information Technology, 2005. pp. 646 –651 (dec 2005)

[52] Mazón, J.N., Trujillo, J., Lechtenborger, J.: Reconciling requirement-driven data warehouses with data sources via multidimensional normal forms. Data & Knowledge Engineering 63(3), 725 – 751 (2007)

[53] McGee-Lennon, M.R.: Requirements engineering for home care technology. In: CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems. pp. 1439–1442. ACM, New York, NY, USA (2008)

[54] McGee-Lennon, M.R., Gray, P.D.: Including stakeholders in the design of home care systems: Identification and categorization of complex user requirements. In: INCLUDE Conference. Royal College of Art, London, UK (April 2007)

[55] Mendes, E.: A systematic review of Web engineering research. In: ISESE'05: International Symposium on Empirical Software Engineering. pp. 498–507. IEEE (2005)

[56] Mens, T., Czarnecki, K., Gorp, P.V.: 04101 discussion – a taxonomy of model transformations. In: Bezivin, J., Heckel, R. (eds.) Language Engineering for Model-Driven Software Development. No. 04101 in Dagstuhl Seminar Proceedings, Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, Dagstuhl, Germany (2005)

[57] Menzies, T.: Editorial: model-based requirements engineering. Requir. Eng. 8(4), 193–194 (2003)

[58] Mittal, K.: Service Oriented Unified Process (SOUP) (2006), http://www.kunalmittal.com/html/soup.shtml

[59] Naslavsky, L., Alspaugh, T.A., Richardson, D.J., Ziv, H.: Using scenarios to support traceability. In: TEFSE '05: Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering. pp. 25–30. ACM, New York, NY, USA (2005)

[60] Object Management Group: MOF 2.0/XMI Mapping Specification, v2.1 (2005), www.omg.org/docs/formal/05-09-01.pdf

[61] Object Management Group: Model Driven Architecture (July, 2001), http://www.omg.org/mda/

[62] OMG (Object Management Group): Meta Object Facility (MOF) Core Specification Version 2.0 (January 2006), http://www.omg.org/cgi-bin/doc?formal/2006-01-01

[63] OMG (Object Management Group): Software Process Engineering Metamodel (SPEM) (April 2008)

[64] Ozkaya, I., Akin, O.: Requirement-driven design: assistance for information traceability in design computing. Design Studies 27(3), 381 – 398 (2006)

[65] Petticrew, M., Roberts, H.: Systematic Reviews in the Social Sciences: A Practical Guide. Blackwell Publishing (2006)

[66] Raj, A., Prabhakar, T.V., Hendryx, S.: Transformation of sbvr business design to uml models. In: ISEC '08: Proceedings of the 1st conference on India software engineering conference. pp. 29–38. ACM, New York, NY, USA (2008)

[67] Ramollari, E., Dranidis, D., Simons, A.J.H.: A survey of service oriented development methodologies, http://staffwww.dcs.shef.ac.uk/people/A.Simons/research/papers/soasurvey.pdf

[68] Rash, J., Hinchey, M., Rouff, C.: Formal requirements-based programming for complex systems. In: ICECCS 2005: Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems, 2005. pp. 116 – 125 (june 2005)

[69] Regnell, B., Runeson, P., Wohlin, C.: Towards integration of use case modelling and usage-based testing. Journal of Systems and Software 50(2), 117 – 130 (2000)

[70] Santos, J., Moreira, A., Araujo, J., Amaral, V., Alferez, M., Kulesza, U.: Generating requirements analysis models from textual requirements. In: MARK '08: First International Workshop on Managing Requirements Knowledge, 2008. pp. 32 –41 (sept 2008)

[71] Shin, J.E., Sutcliffe, A.G., Gregoriades, A.: Scenario advisor tool for requirements engineering. Requirements Engineering 10, 2005 (2004)

[72] Shuja, A., Krebs, J.: IBM®Rational Unified Process®Reference and Certification Guide: Solution Designer. IBM Press (2007)

[73] Sousa, K., Mendonça, H., Vanderdonckt, J., Pimenta, M.S.: Supporting requirements in a traceability approach between business process and user interfaces. In: IHC '08: Proceedings of the VIII Brazilian Symposium on Human Factors in Computing Systems. pp. 272–275. Sociedade Brasileira de Computaç ao, Porto Alegre, Brazil, Brazil (2008)

[74] The SeCSE Team: Towards Service-centric System Engineering, `http://home.dei.polimi.it/baresi/papers/eChallenges05.pdf`

[75] Westfall, L.: Software Requirements Engineering: What, Why, Who, When, and How (2005), `http://www.westfallteam.com/`

[76] Zhang, L., Jiang, W.: Transforming business requirements into bpel: A mda-based approach to web application development. In: WSCS '08: IEEE International Workshop on Semantic Computing and Systems, 2008. pp. 61 –66 (july 2008)

[77] Zimmermann, O., Krogdahl, P., Gee, C.: Elements of service-oriented analysis and design (June 2004), `http://www.ibm.com/developerworks/webservices/library/ws-soad1/`

Appendices

Appendix A

Definitions

This appendix includes general description of concepts which are strongly related with described in this master thesis: methodology concept, necessary elements that have to be defined when specifying a methodology, terms that are frequently used in requirements engineering, terms that are frequently used in model-driven context.

A.1. Methodology

Methodology
In software engineering is a set of rules, principles of methods, suggestions and postulates which should be taken into account in the software development process. These rules are derived on the basis of the systematic study and experience from different types of software projects. It defines methods, procedures or set of them to be applied within different disciplines.

We distinguish two groups of methodologies, classified by the number of rules and their complexity, as well as by the expected effort which demands their use. These two groups are: light (agile) and heavy methodologies.
Depending on the project scope an appropriate methodology should be chosen. Heavy methodologies are not that flexible as the light ones, but serve to manage with huge projects controlling its proper progress regarding to the tasks planning. Heavy methodologies also demand more extensive documentation than those that are light and agile. There are also methodologies that stand in between aforementioned two types. They possess some characteristics from both groups e.g. flexibility in tasks planning with a requisite of a precise large documentation.

Artifact
An artifact is an information used or created during the software development process (in this case it is very often called a *Work Product)*. An artifact can be a formal document, any kind of UML diagrams, implemented functionality or any other software development process product.

Role
A role is a named, specific behavior of a particular unit in a given context.

Activity
An activity is a behavior (action or set of actions) which is done in a given context taking into consideration some input information and which provides also an information as its output.

Task
A unit of work that a role performs. An activity may be decomposed into steps. It is associated with input and output. It is driven by a goal.

Process (workflow)
Process is a ordered sequence of activities also called workflow. Activities are ordered in such a way that output information from one activity can be or even should be reused as an input information for the next activity in the process. These workflows are described by the means of the activity diagrams of UML language.

Discipline
A discipline is a collection of logically related activities and their workflows, artifacts, roles and any other elements such as rules or principles that refer to the software development process.

Iteration
An iteration is a set period of time within a project in which you produce a stable, executable version of the product, together with any other supporting documentation, install scripts, or similar artifacts necessary to use this release. The executable is demonstrable, allowing the team to demonstrate true progress to stakeholders, and get feedback on how they are doing so that they can improve their understanding of what needs to be done and how to do it. Depending on the methodology, an iteration can be longer or shorter, but the time should be regarded as fixed while its content should be adequately managed to meet the schedule. For example in OpenUP it lasts from 1 to 6 weeks.

Phase
A phase is a period of time within the software development process where certain activities from different disciplines should be executed. In the iterative approaches, a phase is a set of iterations. It may consist of one or more iterations depending on the project complexity. The goal of the phase, as a result of execution of the set of activities, is reaching a particular milestone.

Milestone
A milestone is a control point of the project's progress. It aims at providing oversight by raising and answering a set of questions that are typically critical to stakeholders.

Stakeholder
A stakeholder is every person, which has an impact on the project and not belong to the developers team.

A.2. Requirements Engineering

Requirement
A description of a condition or capability of a system; either derived directly from user needs or stated in a contract, standard, specification, or other formally imposed document.

Business requirement
A requirement type which includes information that is related not only to a condition or capability of a system, but also to the business context, organizational structure of the enterprise, processes, etc., which will not necessarily be a part of the system to be developed.

Software requirement
A software requirement is a condition or capability to which a system must conform. Some requirements concern the functionality of a system in the context of actions that the system should be able to perform, these requirements are called **functional requirements**. But there is another type which does not describe functionality of the system, but all these factors that allow to deliver to the user a system to a good quality. These requirements belong to the **non-functional requirements** group.

Feature
Can be treated as a service to be provided by the system that directly fulfills a user need. Because it is easy to discuss these features in natural language and to document

and communicate them, they add important context to the requirements discipline. Sometimes a feature is just another type of requirement.

A.3. Model-driven techniques

Model
A semantically closed abstraction of a system. In the Rational Unified Process, a complete description of a system from a perspective—"complete" meaning that you don't need additional information to understand the system from that perspective; a set of model elements.

Reference Model
It is a model that describe some common terms, a well-defined framework for existing aspects of the specification, or a general overarching structure for a domain. It focuses on interoperability and standardization. A reference model is based on a small number of unifying concepts and is an abstraction of the key concepts, their relationships, and their interfaces both to each other and to the external environment. A reference model may be used as a basis for education and for explaining standards to a non-specialist and can be viewed as a framework for comparing architectures and operations of existing and future systems.

Model Transformation
takes as input a model conforming to a given meta-model and produces as output another model conforming to a given meta-model.

Requirements Traceability
Requirements traceability is concerned with documenting the life of a requirement and to provide bi-directional traceability between various associated requirements. It enables users to find the origin of each requirement and track every change which was made to this requirement. For this purpose, every change made to the requirement should be registered.

Appendix B

OpenUP/MDRE extension summary

This appendix contains the following figures:

- Figure B.1 presents tasks performed by the Analyst (from Model-Driven Requirements discipline) together with related artifacts;
- Figure B.2 presents tasks performed by the Model Analyst (from Model-Driven Requirements discipline) together with related artifacts;
- Figure B.3 presents tasks performed by the Transformation Specifier (from Model-Driven Requirements discipline) together with related artifacts;
- Figure B.4 presents tasks performed by the Process Engineer and by the Infrastructure Specialist (from Environment discipline) together with related artifacts;

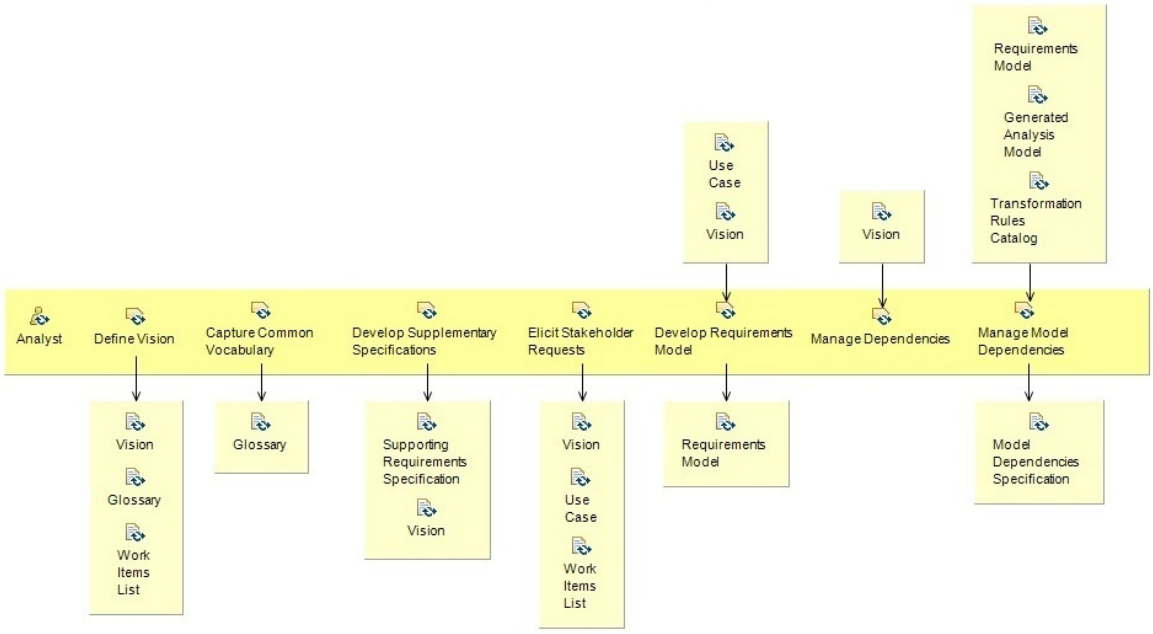

Figure B.1. Tasks and artifacts assigned to the *Analyst* role of the Model-Driven Requirements discipline.

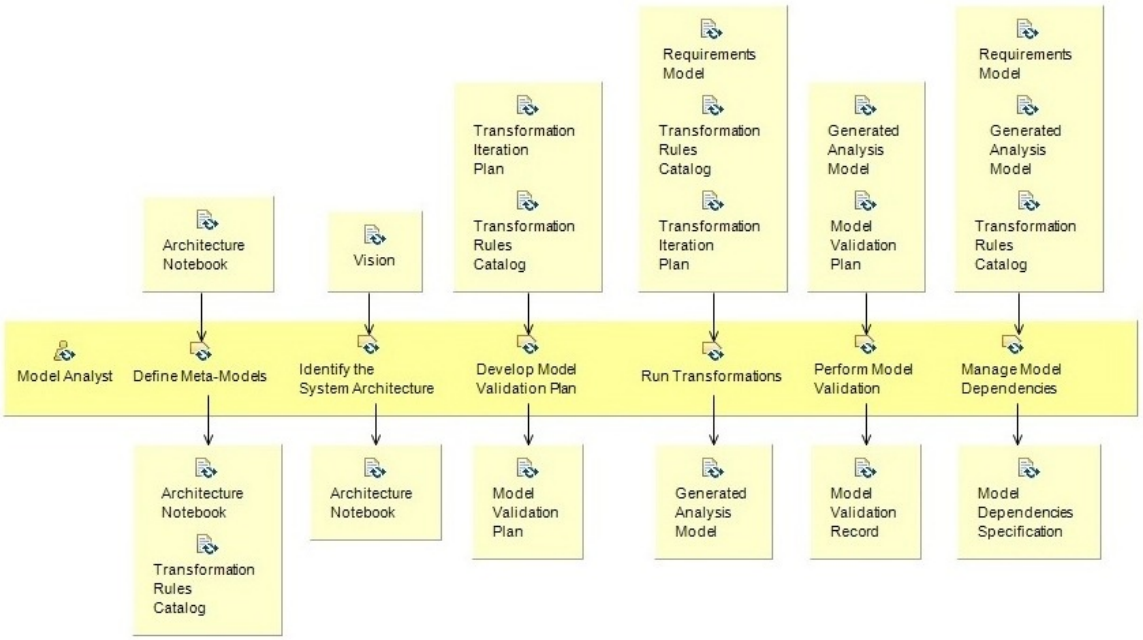

Figure B.2. Tasks and artifacts assigned to the *Model Analyst* role of the Model-Driven Requirements discipline.

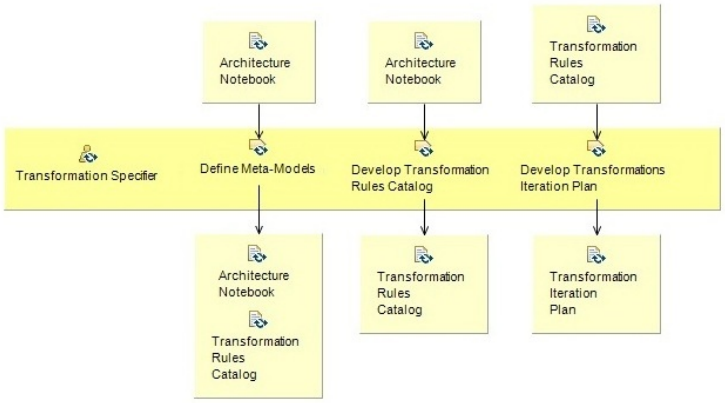

Figure B.3. Tasks and artifacts assigned to the *Transformation Specifier* role of the Model-Driven Requirements discipline.

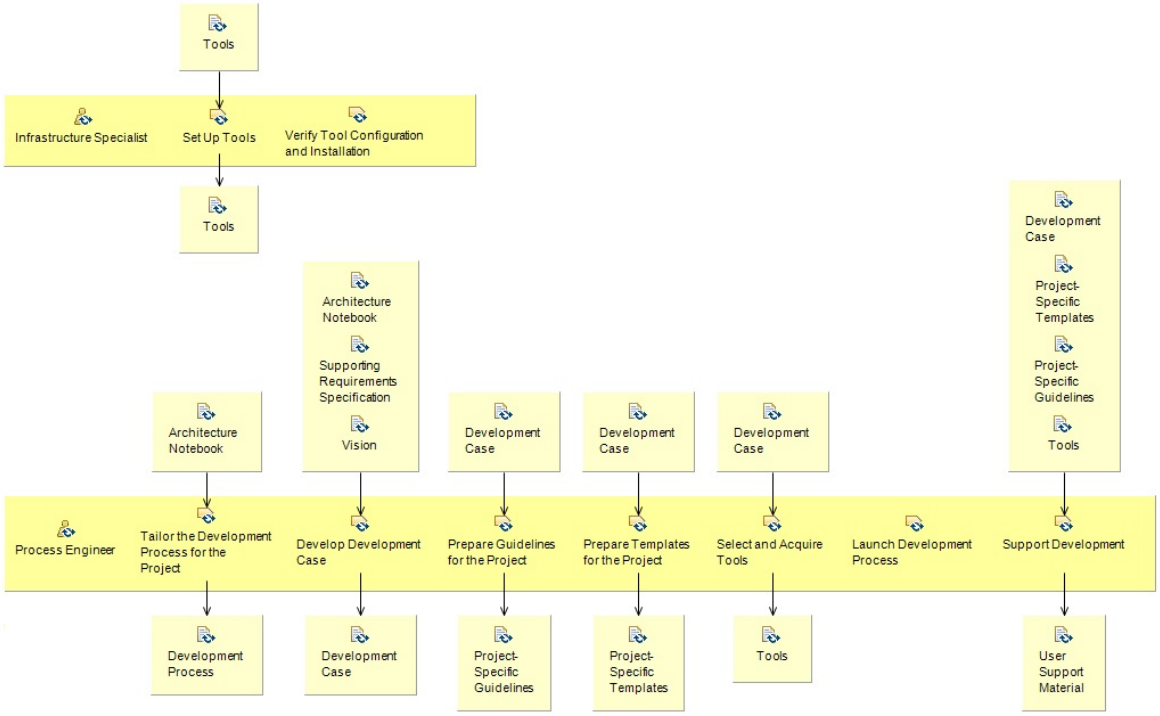

Figure B.4. Roles, tasks and artifacts of the Environment discipline

Appendix C

OpenUP/MDRE definition as EPF plug-in

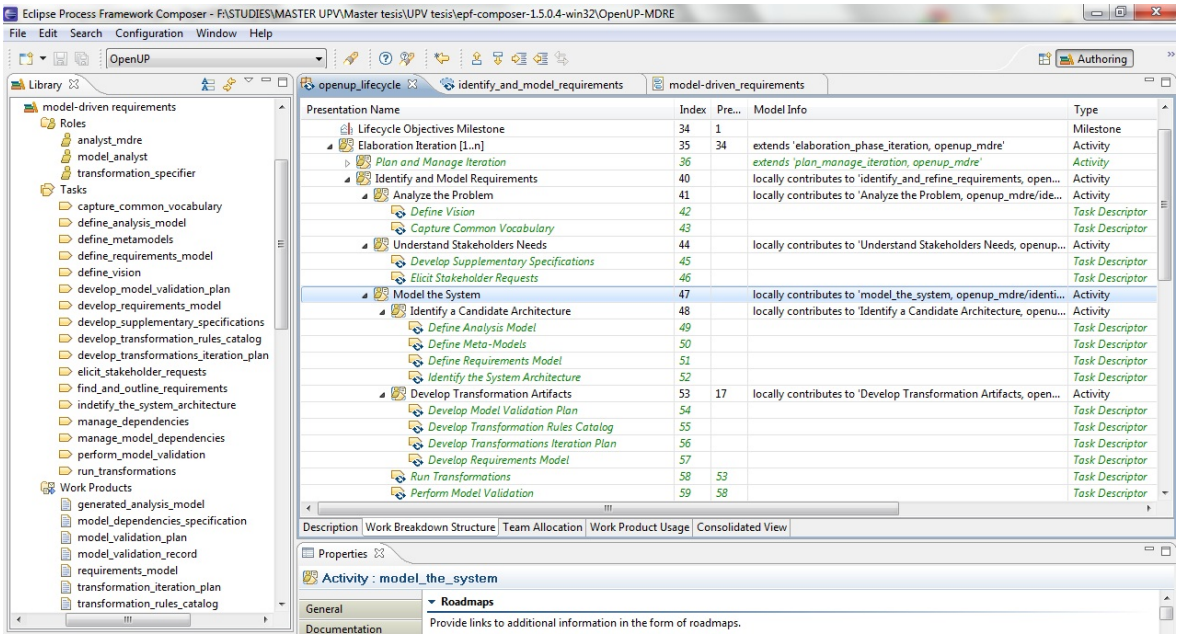

Figure C.1. EPF - Work breakdown structure for *Identify and Model Requirements* activity.

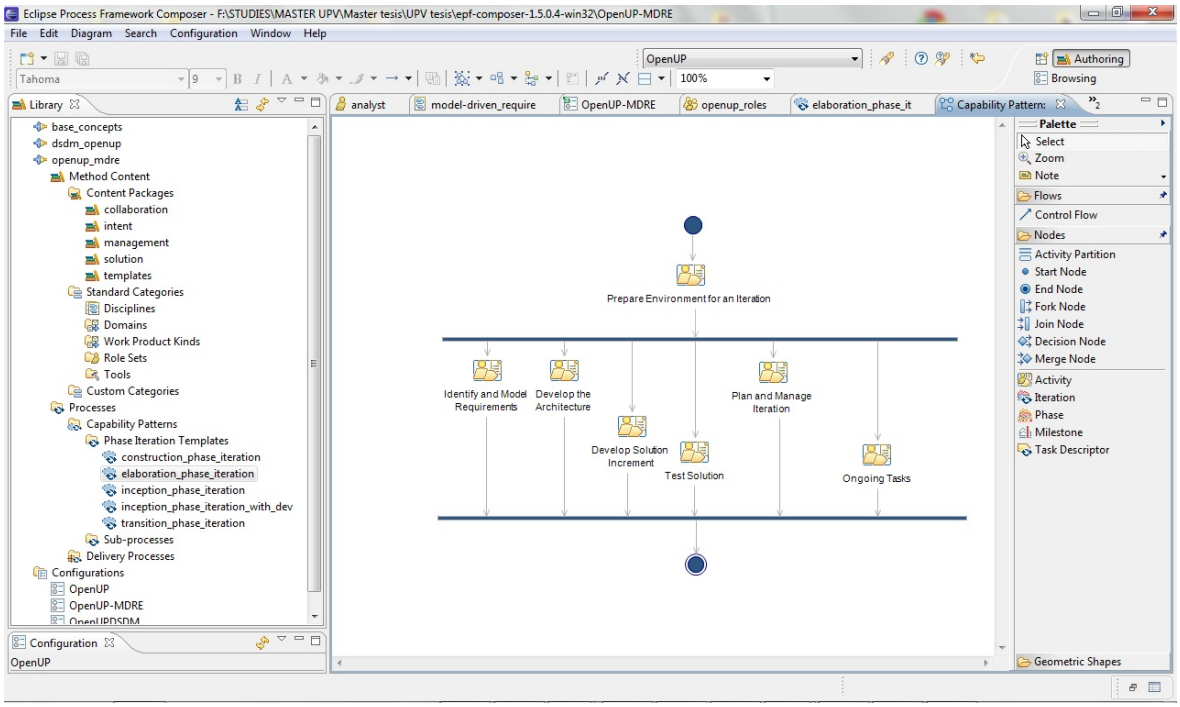

Figure C.2. EPF - Capability pattern modeling for the *Elaboration* phase.

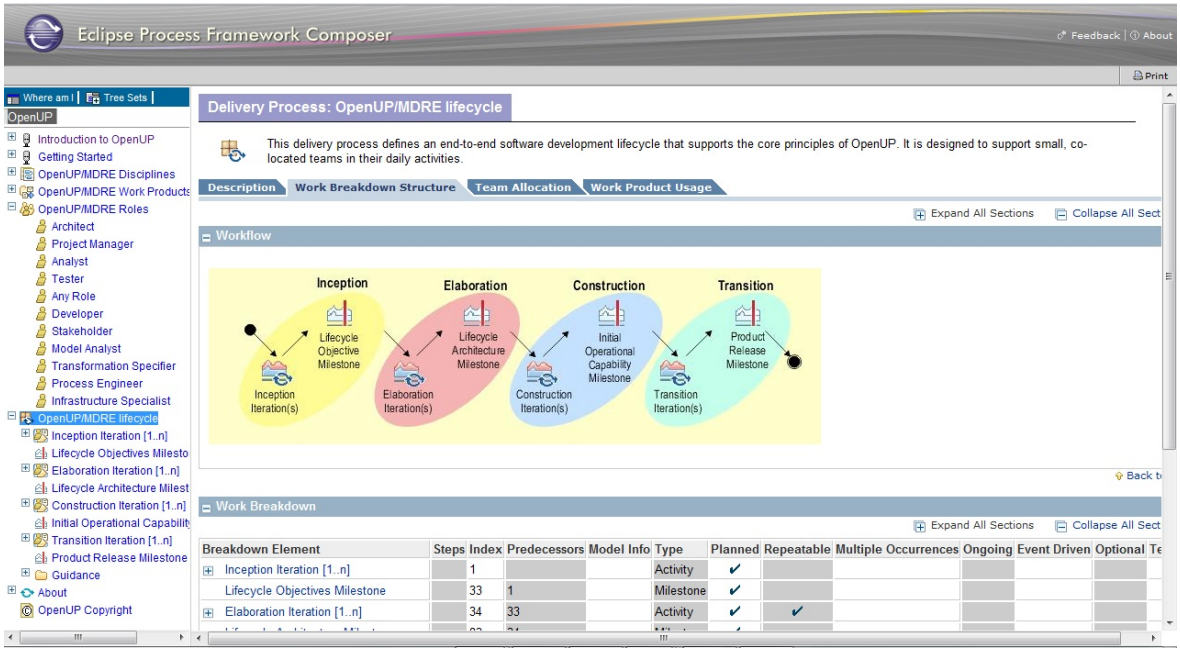

Figure C.3. EPF - published OpenUP/MDRE delivery process.

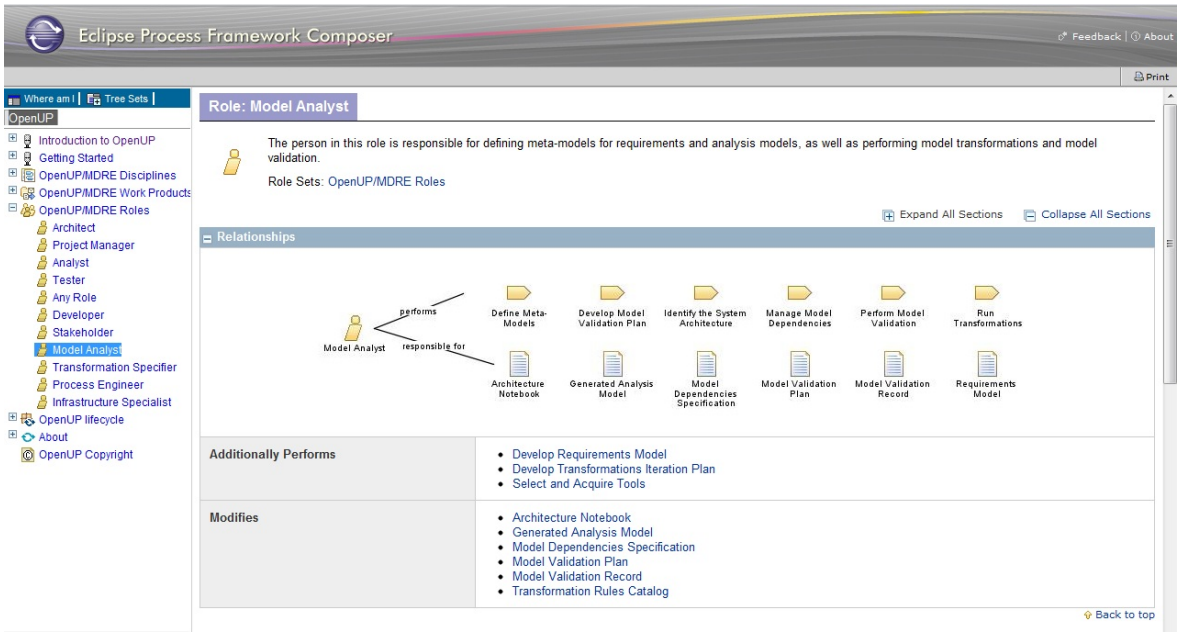

Figure C.4. EPF - published Model Analyst role summary.

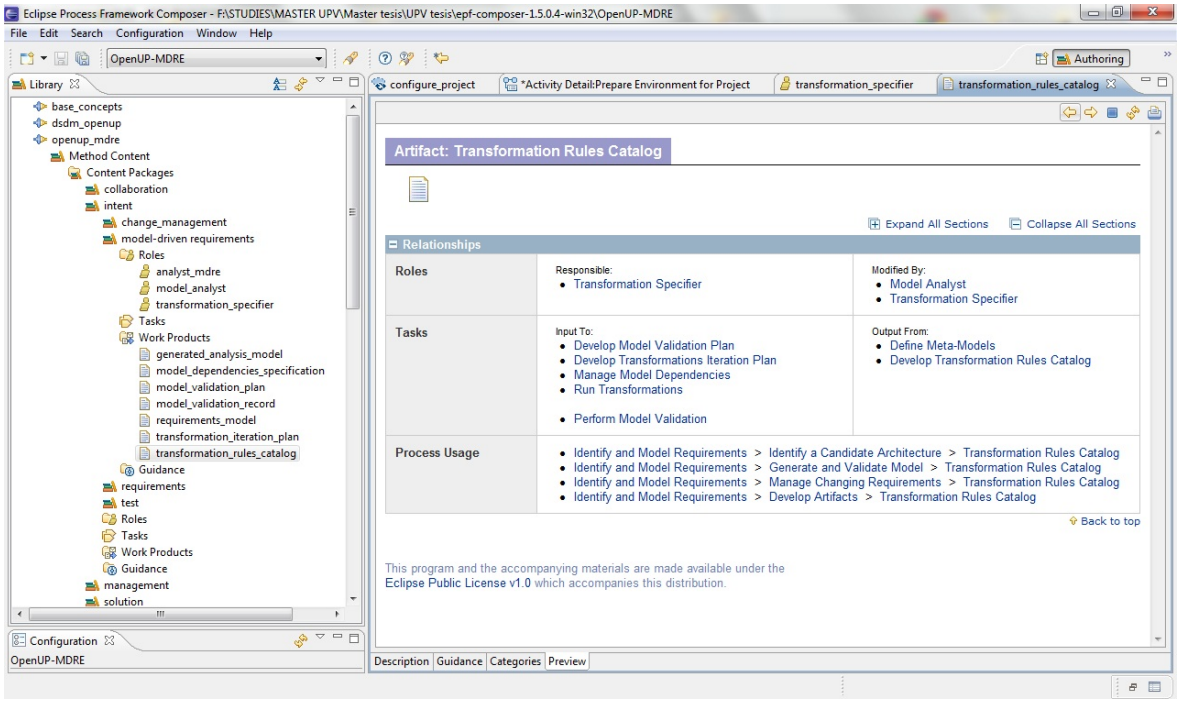

Figure C.5. EPF - Transformation Rules Catalog work product relationships.

Appendix D

Papers included in the Systematic Literature Review

Systematic literature review papers listed in this appendix follow this format:
`author(s): title (year) [bibliographic source]`

1. Samir Kherraf, Éric Lefebvre, Witold Suryn: Transformation From CIM to PIM Using Patterns and Archetypes (2008) [IEEExplore]
2. Chih-Wei Lu, Chih-Hung Chang, William C. Chu, Ya-Wen Cheng, Hsin-Chien Chang: A Requirement Tool to Support Model-based Requirement Engineering (2008) [IEEExplore]
3. Narayan Debnath, María Carmen Leonardi, María Virginia Mauco, Germán Montejano, Daniel Riesco: Improving MDA with Requirements Models (2008) [IEEExplore]
4. Jon Oldevik, Arnor Solberg, Brian Elvesćter, Arne J. Berre: Framework for model transformation and code generation (2002) [IEEExplore]
5. Jean-Louis Boulanger, Van Quang Dao: Requirements Engineering in a Model-based Methodology for Embedded Automotive Software (2008) [IEEExplore]
6. Benoit Baudry, Clémentine Nebut, Yves Le Traon: Model-driven Engineering for Requirements Analysis (2007) [IEEExplore]
7. Chih-Wei Lu, William C. Chu, Chih-Hung Chang, Ching Huey Wang: A Model-based Object oriented Approach to RE (MORE) (2007) [IEEExplore]
8. Riham Hassan, Shawn Bohner, Sherif El-Kassas, Mohamed Eltoweissy: Goal-Oriented, B-Based Formal Derivation of Security Design Specifications (2008) [IEEEXplore]
9. James L. Rash, Michael G. Hinchey, Christopher A. Rouff: Formal Requirements-Based Programming for Complex Systems (2005) [IEEEXplore]
10. Ricardo J. Machado, Joăo M. Fernandes, Paula Monteiro, Helena Rodrigues: Transformation of UML Models for Service-Oriented Software Architectures (2005) [IEEEXplore]
11. David Delahaye, Jean-Frédéric Étienne: Producing UML Models from Focal Specifications (2008) [IEEEXplore]
12. María Virginia Mauco, María Carmen Leonardi, Daniel Riesco, Germán Montejano, Narayan Debnath: Formalising a Derivation Strategy for Formal Specifications from Natural Language Requirements Models (2005) [IEEExplore]
13. Li Zhang, Wei Jiang: Transforming Business Requirements into BPEL a MDA-Based Approach to Web Application Development (2008) [IEEExplore]

14. Ankit Goel, Abhik Roychoudhury: Synthesis and Traceability of Scenario-based Executable Models (2007) [IEEExplore]
15. Christian Kop, Heinrich C. Mayr: Conceptual Predesign Bridging the Gap between Requirements and Conceptual Design (1998) [IEEExplore]
16. CAO Xiao-Xia, MIAO Huai-Kou, XU Qing-Guo: Modeling and Refining the Service-Oriented Requirement (2008) [IEEExplore]
17. Sadaf Mustafiz, Jorg Kienzle, Hans Vangheluwe: Model Transformation of Dependability-Focused Requirements Models (2009) [IEEExplore]
18. Philip Liew, Kostas Kontogiannis, Tack Tong: A Framework for Business Model Driven Development (2004) [IEEExplore]
19. Janis Osis, Erika Asnina: A Business Model to Make Software Development Less Intuitive (2008) [IEEExplore]
20. Joăo Santos, Ana Moreira, João Araújo, Vasco Amaral, Mauricio Alférez, Uirá Kulesza: Generating Requirements Analysis Models from Textual Requirements (2008) [IEEExplore]
21. Renaud De Landtsheer, Emmanuel Letier and Axel van Lamsweerde: Deriving Tabular Event Based Specifications from Goal-Oriented Requirements Models (2003) [IEEExplore]
22. Jane Cleland-Huang, Jane Huffman Hayes, J. M. Domel: Model-Based Traceability (2009) [ACM DL]
23. Stefan Biffl, Richard Mordinyi, Alexander Schatten: A Model-Driven Architecture Approach Using Explicit Stakeholder Quality Requirement Models for Building Dependable Information Systems (2007) [ACM DL]
24. Eric Dubois, Eric Yu, Michael Petit: From Early to Late Formal Requirements: a Process-Control Case Study (1998) [ACM DL]
25. Maritta Heisel, Jeanine Souquiéres: Methodological Support for Requirements Elicitation and Formal Specification (1998) [ACM DL]
26. M.J. Escalona, J.J. Gutiérrez, L. Rodríguez-Catalán, A. Guevara: Model-driven in reverse. The practical experience of the AQUA project (2009) [ACM DL]
27. Alexander Lorenz, Hans-Werner Six: Tailoring UML Activities to Use Case Modeling for Web Application Development (2006) [ACM DL]
28. Nora Koch: Transformation Techniques in the Model-Driven Development Process of UWE (2006) [ACM DL]
29. Alexei Lapouchnian, Yijun Yu, Sotirios Liaskos, John Mylopoulos: Requirements-Driven Design of Autonomic Application Software (2006) [ACM DL]
30. Miguel A. Laguna, Bruno Gonzalez-Baixauli: Requirements Variability Models: Meta-model based Transformations (2005) [ACM DL]
31. Emmanuel Letier, Axel van Lamsweerde: Deriving Operational Software Specifications from System Goals (2002) [ACM DL]
32. F. Duarte, W. Hasling, R. Leao, E. Silva, V. Cortellessa: Extending Model Transformations in the Performance Domain with a Node Modeling Library (2008) [ACM DL]
33. N.C. Narendra, Bart Orriens: Modeling Web Service Composition and Execution via a Requirements Driven Approach (2007) [ACM DL]
34. Salah Kabanda, Mathew Adigun: Extending Model Driven Architecture Benefits to Requirements Engineering (2006) [ACM DL]
35. Kenia Sousa, Hildeberto Mendonça, Jean Vanderdonckt, Marcelo S. Pimenta: Supporting Requirements in a Traceability Approach between Business Process and User Interfaces (2008) [ACM DL]
36. Nora Koch, Gefei Zhang, Mará José Escalona: Model Transformations from Requirements to Web System Design (2006) [ACM DL]

37. Zhi Li: Progressing Problems from Requirements to Specifications in Problem Frames (2008) [ACM DL]
38. John A. Van der Poll, Paula Kotze: Combining UCMs and Formal Methods for Representing and Checking the Validity of Scenarios as User Requirements (2003) [ACM DL]
39. Robert Seater, Daniel Jackson: Problem Frame Transformations: Deriving Specifications from Requirements (2006) [ACM DL]
40. Christian Seybold, Silvio Meier, Martin Glinz: Scenario Driven Modeling and Validation of Requirements Models (2006) [ACM DL]
41. Alexei Lapouchnian, Sotirios Liaskos, John Mylopoulos, Yijun Yu: Towards Requirements-Driven Autonomic Systems Design (2005) [ACM DL]
42. Leila Naslavsky, Thomas A. Alspaugh, Debra J. Richardson, Hadar Ziv: Using Scenarios to Support Traceability (2005) [ACM DL]
43. P. Jamshidi, S. Khoshnevis, R. Teimourzadegan, A. Nikravesh, F. Shams: Toward Automatic Transformation of Enterprise Business Model to Service Model (2009) [ACM DL]
44. Avik Sinha, Amit Paradkar, Clay Williams: On Generating EFSM models from Use Cases (2007) [ACM DL]
45. Joăo M. Fernandes, Simon Tjell, Jens Bćk Jřrgensen, Óscar Ribeiro: Designing Tool Support for Translating Use Cases and UML 2.0 Sequence Diagrams into a Coloured Petri Net (2007) [ACM DL]
46. Amit Raj, T. V. Prabhakar, Stan Hendryx: Transformation of SBVR Business Design to UML Models (2008) [ACM DL]
47. Pedro Valderas, Vicente Pelechano: Introducing requirements traceability support in model-driven development of web applications (2008) [ScienceDirect]
48. Francesco Basile, Pasquale Chiacchio, Domenico Del Grosso: A two-stage modelling architecture for distributed control of real-time industrial systems: Application of UML and Petri Net (2008) [ScienceDirect]
49. Xiaoshan Li, Zhiming Liu: Prototyping System Requirements Model (2008) [ScienceDirect]
50. Jose-Norberto Mazón, Juan Trujillo, Jens Lechtenbörger: Reconciling requirement-driven data warehouses with data sources via multidimensional normal forms (2007) [ScienceDirect]
51. Günther Fliedl, Christian Kop, Heinrich C. Mayr, Alexander Salbrechter, Jürgen Vöhringer, Georg Weber, Christian Winkler: Deriving static and dynamic concepts from software requirements using sophisticated tagging (2007) [ScienceDirect]
52. Ipek Ozkaya, Ömer Akin: Requirement-driven design: assistance for information traceability in design computing (2006) [ScienceDirect]
53. Michael G. Hinchey and James L. Rash and Christopher A. Rouff and Denis Gracanin: Achieving dependability in sensor networks through automated requirements-based programming (2006) [ScienceDirect]
54. Günther Fliedl and Christian Kop and Heinrich C. Mayr: From textual scenarios to a conceptual schema (2005) [ScienceDirect]
55. Björn Regnell and Per Runeson and Claes Wohlin: Towards integration of use case modelling and usage-based testing (2000) [ScienceDirect]
56. Jae Eun Shin, Alistair G. Sutcliffe, Andreas Gregoriades: Scenario advisor tool for requirements engineering (2005) [SpringerLink]
57. Yann Thierry-Mieg, Lom-Messan Hillah: UML behavioral consistency checking using instantiable Petri nets (2008) [SpringerLink]
58. J. Angele, D. Fensel, D. Landes, R. Studer: Developing Knowledge-Based Systems with MIKE (1998) [SpringerLink]

59. Dian-Xiang Xu: Aspect-Oriented Modeling and Verification with Finite State Machines (2009) [SpringerLink]
60. Jane Cleland-Huang, Carl K. Chang, Jeffrey C. Wise: Automating performance-related impact analysis through event based traceability (2003) [SpringerLink]
61. Hung Le Dang, Hubert Dubois, Sébastien Gérard: Towards a traceability model in a MARTE-based methodology for real-time embedded systems (2008) [SpringerLink]
62. Joăo Paulo A. Almeida, Maria-Eugenia Iacob, Pascal van Eck: Requirements traceability in model-driven development: Applying model and transformation conformance (2007) [SpringerLink]
63. E. Insfran, O. Pastor, R. Wieringa: Requirements Engineering-Based Conceptual Modelling (2002) [Manual search]
64. N. A. M. Maiden, S. Manning, S. Jones, J. Greenwood: Generating requirements from systems models using patterns: a case study (2005) [Manual search]
65. Nivedita Deshmukh, Sameer Wadhwa: A Meta Model for Iterative Development of Requirements Leveraging Dynamically Associated Prototyping and Specification Artifacts (2007) [Manual search]
66. Matias Kleiner, Patrick Albert, Jean Bezivin: Parsing SBVR-based Controlled Languages (Empirical) (2009) [Manual search]
67. Javier J. Gutiérrez, Cleméntine Nebut, Maria J. Escalona, Manuel Mejia and Isabel Ramos: Visualization of use cases through automatically generated activity diagrams (2008) [Manual search]
68. Xulin Zhao, Ying Zou, Jen Hawkins, Bhadri Madapusi: A Business-Process-Driven Approach for Generating Ecommerce User Interfaces (2007) [Manual search]
69. Marco Brambilla, Jordi Cabot, Sara Comai: Automatic Generation of Workflow-extended Domain Models (2007) [Manual search]
70. Wei Zhang, Hong Mei, Haiyan Zhao: Transformation from CIM to PIM: A Feature-Oriented Component-based Approach (2005) [Manual search]
71. Martin Giese, Rogardt Heldal: From Informal to Formal Specifications in UML (2004) [Manual search]
72. Shane Sendall, Alfred Strohmeier: From Use Cases to System Operation Specifications (2000) [Manual search]