AUTOMATING SPECIFICATION TO IMPLEMENTATION SOFTWARE DEVELOPMENT USING MODEL DRIVEN ARCHITECTURE

ABD ELGAFFAR HAMED AHMED ALI

UNIVERSITI TEKNOLOGI MALAYSIA

AUTOMATING SPECIFICATION TO IMPLEMENTATION SOFTWARE DEVELOPMENT USING MODEL DRIVEN ARCHITECTURE

ABD ELGAFFAR HAMED AHMED ALI

A thesis submitted in fulfilment of the
requirements for the award of the degree of
Doctor of Philosophy (Computer Sciences)

Faculty of Computer Science and Information Systems
Universiti Teknologi Malaysia

FEBRUARY 2011

To Professor Sheik Hassan (Blessings be upon him), who has changed my life.

To his beloved son Sheik Mohammed

For my great mother and father

To my supervisors and sponsor

ACKNOWLEDGEMENT

Praise to the Almighty, who has provided whatever talent I might have, and has given the great gifts which have helped me to reach this point.

Professor Sheik Hassan ElFatih(blessings to him who passed away) my first beloved teacher, inspired me to loyalty and wisdom . He taught me the origin of ethics and how to practice it. I belief it was the main reason of any success I have it in my life generally and for the Phd especially. Finally the one who teach me you do not thank Allah if you do not thank those who helped you.

My beloved person his son Sheik Mohammed he always guide me by his wise advices. His concern and taking care of study let me continue my postgraduate study and award me the progress I have in my field of research. It has been of significant value. I wish to thank his unflagging support during this period.

Professor Shamsul Bin Sahibuddin for stimulating suggestions and encouragement helped me during my study years at UTM, and from him I received a great balance of freedom, monitoring, and guidance in gaining a challenging yet enjoyable learning experience of doing research.

Professor Robert Colomb who suggested the original topic for my thesis and who helped me in many ways to develop it. His advice on both the thesis research and on how to behave as a professional has been of enormous value.

I would like to pay tribute to Sudan University of Science and Technology for sponsoring my PhD study.

My great appreciations for my parents who struggle during my study keep me stable and finish my study on time. Also I would like my brother Habib Alawi for his collaboration and help.

ABSTRACT

Model driven architecture (MDA) is a new development methodology which raises abstraction and re-using levels. MDA is aimed at developing applications in established domains without writing new codes. To this end, models are first-class artifacts where the specification of the system is modeled using platform independent model (PIM). The implementation is modeled as code-based API using platform specific model (PSM). MDA is about mapping PIM to PSM, whereby specifications will translate into calls to the code-based API which execute it. The ultimate goal of MDA is to automate this process. This process is not specified in detail in standard Object Management Group (OMG) document. Due to lack of previous work tackling the development problem from specification to implementation, this research proposes End to End Development Engineering (E2EDE) method using MDA methodology. E2EDE is a novel approach to software engineering, where the notion of variability is utilized from Software Product Line and used to model design decisions in PSM. PIM is equipped with Non-Functional Requirements (NFRs) to inform design decisions; thereby guiding the mapping process. A Unified Modeling Language (UML) profile is developed for modelling NFRs in PIM and Meta-Object Facility (MOF) profile for modeling variability in PSM. To address mapping variability and its modeling, an MOF metamodel is developed. In addition, a strategic PSM is developed for messaging systems to be configured into different applications such as a helpdesk system. Two different case studies with different scales are used to evaluate E2EDE. Finally, Profile UML package and model manipulation approach is taken to implement aspects of E2EDE. Being applied in different cases, the E2EDE has shown productivity by allowing reuse of different artifacts: PIM and PSM mapping metamodels and also encouraging mapping automation.

ABSTRAK

Seni bina pacuan model (MDA) adalah metodologi pembangunan baru yang meningkatkan tahap peniskalaan dan guna semula. MDA disasarkan pada pembangunan aplikasi bagi domain yang mantap tanpa menulis kod baru. Untuk tujuan ini, model merupakan kelas pertama yang mana spesifikasi sistem dimodelkan menggunakan model platform bebas (PIM), sementara pelaksanaannya dimodelkan menggunakan model platform spesifik (PSM) sebagai model kod berasaskan API. MDA adalah pemetaan PIM ke PSM yang mana spesifikasinya akan diterjemah kepada panggilan ke kod-asas API yang akan melaksanakannya. Matlamat akhir MDA adalah untuk mengautomasikan proses ini. Proses pengautomasian ini tidak ditentukan secara terperinci dalam dokumen piawai Object Management Group (OMG). Oleh kerana kurangnya kajian terdahulu yang menangani masalah pembangunan daripada spesifikasi hingga pelaksanaan, kajian ini mencadangkan kaedah kejuruteraan Pembangunan Hujung ke Hujung (E2EDE) menggunakan metodologi MDA. E2EDE adalah pendekatan baru dalam dunia kejuruteraan perisian, yang mana idea kepelbagaian diambil dari Barisan Produk Perisian dan digunakan untuk membina model reka bentuk keputusan dalam PSM. PIM dilengkapkan dengan Keperluan Bukan Fungsi (NFRs) untuk memaklumkan mengenai keputusan reka bentuk dan seterusnya membantu proses pemetaan. Profil Unified Modeling Language (UML) dibangunkan untuk membentuk NFRs dalam profil PIM dan Meta Object Facility (MOF) dalam membentuk kepelbagaian dalam PSM. Metamodel MOF dibangunkan untuk memudahkan kepelbagaian pemetaan dan pemodelannya. Selain itu, PSM strategik telah dibangunkan bagi sistem pesanan yang dapat dikonfigurasikan bagi menghasilkan aplikasi berbeza seperti sistem meja bantuan. Dua kajian kes dengan skala yang berbeza digunakan untuk menilai E2EDE. Akhir sekali, pakej UML Profil dan pendekatan manipulasi model diaplikasikan bagi melaksanakan aspek-aspek E2EDE. Setelah digunakan dalam kes yang berbeza, E2EDE menunjukkan produktiviti dengan membolehkan penggunaan semula artifak yang berbeza iaitu PIM dan pemetaan metamodel PSM dan juga menggalakkan automasi pemetaan.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

API	-	Application Programming Interface
CASE	-	Computer Aided Software Engineering
COTS	-	Commercial Off-The-Shelf
CORBA	-	Common Object Request Broker Architecture
EMF	-	Eclipse Modeling Framework
FIPA	-	Foundations of Intelligent Physical Agent
FITA	-	Federation of International Archery
OMG	-	Object Management Group
IEEE	-	Institute of Electrical and Electronics Engineers
ISO/IEC	-	International Organization for Standardization/ Engineering Consortium
MOF	-	Meta Object Facility
J2EE	-	Java 2 Platform Enterprise Edition
MDA	-	Model Driven Architecture
MOF	-	Meta Object Facility
MQ	-	Message Queuing
MSMQ	-	Microsoft Message Queuing
OCL	-	Object Constraint Language
OMG	-	Object Management Group
ODM	-	Ontology Definition Metamodel
PFA	-	Product Family Architecture

PIM	-	Platform Independent Model
PSM	-	Platform Specific Model
WMC	-	Workflow Management Collation
SQL	-	Structured Query Language
QVT	-	Query/Views/Transformations
UML	-	Unified Modelling Language
UML2	-	Unified Modelling Language, version XML eXtensible Markup Language
XMI	-	XML Metadata Interchange

CHAPTER 1

INTRODUCTION TO THESIS

1.1 Introduction

"An important aspect of some of the latest OMG standards is that they greatly advance the art, science, and scope, of modeling. The combined power of these model-driven standards forms the basis of a compelling approach to long-lived architectures for distributed, component-based systems." (OMG, 2003).

Today a majority of software development strategies require complexity because of code-based adoption and manual efforts that make the cost of their products high. Requirements are increasing and changes from time to time lead to inflating the total cost. The craft of code has become tedious until it begins to look like surgery operations (due largely to increasing demand for high quality). On one hand, developers rely entirely on code so they express the model of the system they are building in a third-generation programming language like java, C#, or C++ (i.e. packages, modules), and any model for architecture designs are left separate and informal. On the other hand, the evolution of these solutions becomes unavoidable. For instance, the technologies are volatile even within the same technology paradigm; versions produced with different capabilities are shipped by software providers continuously. This situation complicates the understanding of the system in general and makes it difficult to manage the evolution especially in large scale systems. Therefore, the investments in systems based on this traditional methods and the instability of technologies and requirements, yield disappointing results because many projects fail and others go over budget.

The opening quotation is by the Object Management Group, a non-profit organization that embraces about 800 of the largest companies in the IT world. It is developing many of the dominant standards in industry, in particular UML. This thesis is about a new trend for software architecture whereby a *model-driven* approach is followed to guard against the effects of changes in software technology. The OMG's version of this model-based approach is called Model-Driven Architecture, or MDA (OMG, 2003).

One reason behind MDA that is because the ability of current software development artifact to adapt is limited so longevity of artifacts is short. As systems are growing and becoming very complex, the efforts of developing and modifying them with code-based approaches are more expensive, even sometimes impossible which leads to software failures. This means the development artifacts are required to be of high quality and deliver more functionality to cope with such challenges. On other hand, the ability to have strategic software assets (i.e. models) is needed to reduce the high cost through the notion of re-use. Less human intervention or increased automation is a proven factor for increasing productivity by removing complex tasks and mechanizing the process of producing code.

It turns out that the extent to which quality, longevity, and cost of production are balanced when developing software determines the economic viability of the development strategy (Frankel, 2003).

Modeling and models have a high impact on communication among humans which helps manage complexity and is a means by which a system is documented.

Raising the abstraction level and re-usability are the key success factors for building software that is not only reliable but also economically viable. This concept is what is meant by the the above OMG quotation. As a consequence of abstraction reuse increases also, because abstraction from the OMG viewpoint is based on language and has well defined form ("Syntax") and meaning ("semantics"). Hence, developing software by this new adopted approach is considering Models or specifications as first class entities. This is why this approach is called Model Driven Architecture (MDA).

There are longer established re-use approaches such design pattern (Gamma et al., 1995) and software product line (Pohl et al., 2005) in the literature. The former introduces the concept of design re-use whereby recurring pairs of problem and solution is prepared to be reusable any number of times for any similar class of problems. The latter is about finding commonalities among systems to identify components that can be applied to many systems, and to identify program families that are positioned to take fullest advantage of those components (Weiss et al., 1999). This process is called domain engineering which is a matter of variability management.

Design Pattern raises abstraction to a higher level than object in the object programming model, because it consists of many objects collaborating to fulfill some responsibility such as Publish-subscribe in a distributed system. Software product line increases the abstraction level and re-using theme even higher than design patterns. In fact components might include a number of design patterns based on their complexity. Furthermore, software product line is built around the concept of artifact re-use such as requirements models, architecture etc. Therefore both are forms of raising abstraction levels and the theme of re-use.

MDA is more recent, following the same trend but in a more effective and economical model-driven way. MDA allows the same model to be realized on multiple platforms thereby improving portability. It improves integration based on models of relationships across different domain applications, and standardizes component interfaces thereby affording *interoperability*. It increases *reusability* by embracing standard platforms for example CORBA, XML, J2EE.

Therefore MDA is the most promising re-use approach, but for several reasons the relationship between MDA and the other longer established re-use disciplines should be studied. This is why in this thesis exploring and investigating the relationship between MDA, software product line and design pattern is pivotal. As consequence of that I will show how MDA can build on both of the earlier approaches.

1.2 Motivation

"*The MDA defines an approach to IT system specification that separates the specification of system functionality from the specification of the implementation of that functionality on a specific technology platform. To this end, the MDA defines an architecture for models that provides a set of guidelines for structuring specifications expressed as models.*" (OMG, 2003).

From the above OMG text we observe three key concepts: specification, separation of concerns (business functionality-implementation), and architecture of models.

Due to increasing software requirements and resulting complexity in current software, effective software development methods become essential. The automation from specification through to implementation is a long-standing aim in software engineering. Typically current software we use and build exhibits increasing variability for example in business rules, and the technology which changes within even the same technology paradigm. As a result there are a number of versions shipped to customers. For example java has a number of versions like JDK 1.4 and at the time of writing it had reached java 7. The cause of this variability includes the pressure to add new features or enhance the functionality of service provided by the system and non-functional aspects. Therefore it becomes difficult to manage the cost as there are great resources needed to cope with these changes. Adding new features to a legacy system would require checking the integration between system parts which is considered a challenging task in the code-based approach. If we look at the current demand and size of software, we could conceive the situation :

1- "*The size, complexity, and scope of service-oriented architecture continue to grow as these systems are starting to cross organizational boundaries and a market of third-party services is emerging...*"(Bianco et al., 2009)

These include in particular eBay- like companies offering platforms for managing bids, generating commissions, and performing other functions related to online auctions, and Amazon-like companies offering services for businesses, consumers, associates, sellers and others.

2- Google Chrome (2010) has a new generation of browsers characterized by the highest security, working differently from browsers like Mozilla and Internet Explorer but having same functionality. The reason is to add more security, performance and reliability.
3- Two universities such as UTM and SUST for example could agree on running shared postgraduate programs where students could study partially in one university and complete their studies in the other. They share resources such as credit hours and etc. So their credit hours studied at either university should be kept for awarding the degree.

In fact these are only a few of a large number of examples for large, medium and small scale software but these few can help us see a picture of this reality. So the main point here we can see through a number of questions:

How to extend the traditional methods (Code-based) in a long-lived architecture to deliver these new businesses? And *how to provide an effective integration with* *legacy systems? How much change is needed to incorporate these requirements? How long will this process take? If there is new features needed to be added is it easier to add them? If a decision is made to change technology (acquiring new quality such as security and performance) is the design easily adaptable?* And so on.

MDA is a new development trend intended to make it possible to develop applications in established domains without writing program code. It considers models as first class objects, where the development process is driven by models. Using UML or MOF language (is a subset of UML and new standard language has more specification power), the business functionality is specified with business concepts at a high abstraction level. This application-oriented language is called platform independent model (PIM). Because we have an established code base that implements large classes of applications, such as relational databases, a MOF model is developed for the application program interface (API) of that code base. This model of API called a platform specific model (PSM) of which there might be more than one. The mapping between business-oriented modeling language and the code base API is established by using a MOF-based tool like Query View Transform (QVT). The result of the mapping is that the business specification will be transformed into calls into the code base that execute it.

The PIMs and PSMs are developed independent of specific applications, and mappings are also specified independent of PIM or PSM so that they form the assets in the software re-using discipline. Therefore, since the application has been specified, the code is generated automatically thereby leveraging the investment in model assets.

MDA is enabled by models that can be formalized and which are machine readable. Formalization means removing ambiguity with well-defined syntax and semantics. This is the foundation for automation, the ultimate goal.

MDA is intended to be delivered by languages (programming languages) in software engineering, so it has been recognized in the community (this is discussed at length in Chapter 4). As is said :

" *Software language should indeed be considered as a single coin but with two sides*: one side for the machine and the other for the software engineers" (Favre et al., 2009).

In fact many languages are needed in the software production life cycle for different purposes, such as specification, testing, constraints etc. Therefore a family of languages currently is needed to be provided in the development environment.

The value of having a common language besides the clarity, reusability, and efficiency of its execution, is a standardized process offering an additional safeguard against miscommunication and waste (Rasmussen and Niles, 2007).

From these points we conclude that a software system becomes an information system in itself where different assets and artifacts need to be properly managed with capability of mappings between models, synchronization and production of code. More importantly while integrating artifacts during product-life cycle re-use of these assets should be provided. To this end, MDA has followed the information system technology to afford these capabilities.

It was recognized that to make MDA possible would require the development and delivery of a number of modeling and model-manipulation tools. OMG has crafted already some of them as specifications implemented in a number of products, the Meta-Object Facility (MOF), UML, and others. These specifications undergo continuing improvement resulting in MOF 2 (OMG, 2006a), UML 2 (OMG, 2007b).It was recognized that a number of new specifications needed to be formulated and agreed, including Object Constraint Language (OCL) (OMG, 2006b) XML Metadata Interchange (XMI) (OMG, 2003a) and Query/View/Transformation (QVT) (Steinberg et al., 2008), among others. In addition, toolset of platforms are needed, some of which available as open source like the Eclipse Modeling Framework (EMF) (Czarnecki and Helsen, 2003) while others are commercially provided by vendors, like IBM MDA products.

1.3 Problem Statement

Previous responses in the context of MDA focus on filling the components and infrastructure gap such as the transformation languages (i.e. QVT, explained in Chapter 4 (Czarnecki and Helsen, 2003) and toolset to automate the mapping process and support of CASE tools (Frankel, 2003) . A few attempts were found describes how to fulfil the gap of end to end engineering which afford developing applications entirely from specification to implementation without writing code. Finding a concrete mapping methods under MDA principles is still a research question. This the fact that although the framework is established by OMG but still how it is to be operationalized is left unspecified, subject to research and experiences of practice. Therefore the questions tackled by this thesis are:

1) **How can automation be accomplished in an end to end development of software from specification to implementation?**
2) **How variability can be incorporated in the end to end software development?**

The concept of variability has emerged in the domain engineering whereby software artifacts can be customized or adapted to produce different sort of kinds. There is of variability that exists on a specific domain and/or the mappings itself. This variability can be classified mainly into structural and behavioral. The former gives different implementation alternatives while the latter gives different process execution paths. For example inheritance representation in the relational model follows structures (i.e. two-tables approach, single-table approach as discussed in Chapter 7,8). An example of behavioral variability is a database transaction, of which there are several kinds, including two-phase commit transaction and the familiar database transaction. The former has an overhead process to achieve reliability and consistency though it has a different process flow.

Recently the relationship between software product line and MDA has been studied. They share a body of knowledge which would help guide us in solving the problem at hand, under the general observation they are working under the umbrella of re-using.

The functional requirements are that what system should perform. Non-functional requirements specify overall characteristics such as cost and reliability. This should be contrasted with functional requirements as defined in requirements engineering, which specify particular behaviors of a system. For example, displaying the number of records in database is functional but how it will be kept up-to-date is NFR. NFRs are hard to measure and determine. The representation and categorization of NFRs shows diversity in terminology and orientation (Matinlassi, 2006). Another example of NFRs is the number of transaction for a specific table in schema. The sort of NFRs relating to architecture is known as quality attributes.

The influence of the quality attributes in software design as a major development cost is recognized in the research recently such as (Zhu and Ian, 2007) (Korherr, 2007) . Modeling these nonfunctional requirements in software architecture design is still under research. Nonfunctional requirements are critical, but they are difficult to specify, model and categorize. Sometimes they conflict with each other. They also sometimes overlap. NFRs by their nature sometimes affect functionality and because of this it is difficult to NFR and functional requirements. Generally this field is considered among a less developed one.

Furthermore, the relationship between nonfunctional requirements and design decisions under MDA principles has not been fully addressed. To our knowledge this thesis is the first direct attempt. It is not recognized how could be represent explicitly, and how to resolve conflicts at some times NFRs contribute to design decision but it can be in conflict with them at another time. A few attempt was done at the level of software architecture such as in (Bass et al., 2003)(Booch, 1991).

In fact, MDA structure more or less contributes to the addressing of NFRs in its models while also allowing generalization of design decisions. For example, portability and re usability is inherited from the separation of concerns concept. Separation of concerns means having a conceptual model view and implementation view separated from each other. In addition, built in mechanisms of extension and specialization such as Profile and OCL, and the MOF metamodel can allow us to develop a dialect or family of languages to add any kind of information either to PIM or PSM, for example NFRs or general design decisions.

As a response to these issues E2EDE has emerged as an effective software development method aimed at automating the craft of code using MDA. MDA adopts abstract computation in which the functionality of the system is specified in PIM metamodel, for example UML class model, and the implementation interfaces for existing bodies of code is specified as PSM metamodels using MOF-based language. MDA is used to map the specification end to the implementation end using a transformation language (model-to-model) such as QVT and eventually to the code (model-to-text). The mapping process which routinely occurs is expected to be done automatically at the model instance level.

Finally, MDA is a young discipline so still there is a question on how to develop a program in this context. In particular, E2EDE is about filling the mapping gap between PIM and PSM.

1.4 Objective

This research gives an answer to the research question by providing a model for E2EDE automation. The objective of the research:

- **To model Variability in PSM**
- **To model Non-functional requirements in PIM**
- **To establish a suitable representation for variability, Non-functional requirements and mappings**

The proposed model involves a number of activities in E2EDE enabling the transformation process.

Two case studies are selected to evaluate the model. The first is intended to understand nonfunctional requirements better while the second is for variability aspects. Generally two common domains in industry and academia are investigated by the two case studies.

1.5 The Research Scope

The E2EDE engineering is intended to provide generic design decisions from implementation space point of view for End-to-End program development. The alternatives for design decisions in the PSM are the main interest. In fact E2EDE can be used for two arbitrary ends such as mapping Object model (PIM) to relational model (PSM). The approach basically considers nonfunctional requirements as guidance to the selection of design decisions that help construct the PSM model instance. Therefore the research focuses on a family of applications with variability in implementation aspects affected by nonfunctional requirements. Therefore it considers the body of variability knowledge relevant to that problem, for example the type of variability explicit to the PSM. In addition, mapping variability is also an important aspect of E2EDE.

Because our concern is on specification to implementation under the MDA context this research study does not include the second part of MDA from implementation to physical code which is a different class of problems. Our study assumes that the PSM is an interface to an existing body of code.

1.6 Contributions

The principal contribution of this work is an End to End Development engineering methodology for realizing the generating of implementation models from specification models automatically. This is contributing to the mapping gap from PIM to PSM in MDA. This will give an answer to the question of *how to write a program in MDA*.

Practically this study will generate concrete products contributing to the MDA context such as Profiles, metamodels. Concretely, the technical knowledge and guidelines is presented for two common domains (Chapters 7 and 8).

The following is a summary of the thesis contributions:

1. An explicit representation of variability in design decisions at PSM metamodel

Chapter 5 motivates the need for inclusion of an activity to specify design decision visibilities in E2EDE software development process. Design choices that expose different qualities are identified and represented using a context free language. This presented in Chapter 7.

Documenting design visibilities provides a systematic way for representing PSM's construct alternatives so contributing basically to the mapping process.
This study has proposed a variability analysis where a classification is proposed, called nonfunctional variability, which is discussed in Chapter 7. In addition, classification from literature are behavioral and structural.

A secondary benefit is to focus early on software quality as this becomes a critical point for most of the current software systems. This is especially critical in ultra-large systems and with the increasingly common situation of technology variability that rapidly changes because of enhancing functionality (e.g. presenting an application on multiple platforms) or concern with nonfunctional aspects (i.e. Google Chrome).

To this end, software architect with contributions 1 and 2 can tell about specific model elements and reason about software architecture thereby rationality is supported.

2.An explicit representation for Nonfunctional requirements at PIM metamodel that can guide the selection of alternative transformations and a high quality design artifact represented by PSM metamodel.

Chapter 7 studies nonfunctional requirements to be specified in the conceptual model in an E2EDE software development process, which considers representation and prioritization of NFRs. But before that Chapter 6 presents a background from literature on how nonfunctional requirements are treated, including definitions, identification, classification and their relationship with functional aspects.

A profile is presented in Chapter 7 as a formal technique for modeling NFRs at PIM metamodel. This technique is a lightweight approach (i.e. profile) of extension to the UML metamodel. This consideration of NFRs early will facilitate the mapping automation between PIM and PSM, but at the same time adds value for selection among alternatives of transformation strategy (mapping variability), which will be based on the desired quality properties of the resulting model.

A new classification is identified by this study to NFRs, namely Package level and class level. Both are discussed in Chapters 7 and 8.

However this technique provides the software engineer with a means for explicit description and reasoning about alternative deigns and transformations so that the software engineer is able to identify the transformations that produce a result with certain quality properties.

A context free language for NFRs using UML profile is presented as solution in chapter 7.

3. Mapping variability and a metamodel for its representation

In Chapter 2 mapping is studied and considered in the broader context of Model Driven Engineering. The details include the meaning of mapping variability and how it can fit in development lifecycle. The study includes a different approach to the implementation of transformation languages such as QVT.

The mappings variability is first introduced in Chapter 7 then in Chapter 8 is exemplified through case study the formulation of requirements that E2EDE must fulfill to provide an adequate support for end to end engineering. An Object model to Relational Model is given as example which also helps show mapping variability.

A dialect for variability is developed in Chapter 7 using MOF profile.

4. A package, profile representation and model manipulation as a mechanism for implementing E2EDE.

Chapter 7 presents a discussion of a practical approach toward implementing E2EDE. Current techniques to implement MDA are based on EMOF, XMI and QVT tools for editing, rendering and mapping metamodel. We propose in this approach a UML package mechanism and profile representation that can be manipulated easily by metaprograms. However, we present a general model for practical solution under E2EDE principles so it can easily be extended into a working system.

This approach overcomes problems experienced in extracting NFR from PIM metamodel where NFR instances are specified at different abstraction levels, resolving the conflict between Package level NFR and Class level NFRs through priority mechanism and aggregating related NFRs or variation points by introducing a Guard predicate. Finally it presents a method to select suitable PSM variants at variation points using NFR instances in the PIM.

5. A strategic standard PSM for Messaging systems.

The realistic nature of our approach, especially the generation of standard PSMs for existing current platforms is discussed in Chapter 8. Also we discuss the issue of the ability of re-using PIM and PSM and mappings if they reach stability with their new kind of knowledge of NFRs and variability. We have investigated a number of applications under a messaging system domain. The result was a PSM metamodel developed that can be re-used to produce different products in that domain. This is presented in Chapter 7. Of course more refinement to this standardized PSM is needed to complete it. The value of re-use in the software development is studied in Chapter 2 and Chapter 8.

1.7 Outline of the thesis

Chapter 1 and 2 provide an introduction to both thesis and review of literature. While Chapter is intended to expose explicitly the research method adopted in this thesis.

Chapter 4 provides a model driven architecture framework and the state-of-the-art to put E2EDE in context. It starts out by the concept of raising abstraction level and re-use and tracing that back in computing history. Then important concepts and terminology that are necessary to understand MDA are introduced. The important MDA mechanisms and technologies such as MOF are explained and demonstrated by examples. Finally the design tradeoffs between the two common extension mechanisms in MDA: Profile and metamodel, which are needed in chapter 7 and 8, are clarified. QVT standard mapping language is introduced. It starts with example of mapping. It discusses why languages generally are a new trend in the software community. After that, more technical details are give about QVT such as part of its essential abstract syntax and concrete syntax based on the OMG QVT standard with examples In addition, it identifies the input, output and how it looks like. It concludes with exposing anatomy of how MDA works and defines the art behind that.

Chapter 5 describes the Software product line as major competitive re-using approach to MDA by introducing its main concepts. The domain engineering concept has been given more concentration because it is the key idea behind software product line. To this end, the variability management mechanisms which realize the software product line are provided. After that, it investigates the relationship between MDA and software product line and how MDA can fit in software product line. At the end it presents taxonomy of variability and the body of knowledge making variability applicable in the MDA context is canvassed.

Chapter 6 explains a second re-using approach, Design pattern, by describing the main concepts and pattern life cycle. It then presents comparisons between MDA and Design pattern and how MDA can fit into Design pattern. Finally it deals with nonfunctional requirements which was observed embedded with design pattern alternative discussion. A nonfunctional requirement is an important concept to an E2EDE so it is applicability in the MDA context is presented at the end.

Chapter 7 describes the basic contribution, the idea of the E2EDE approach. The design issues are an important component so are introduced first. This includes the demonstration of developing two profiles, one for nonfunctional requirements and the other for variability. Then the implementation aspect of E2EDE is explained. To this end metamodels as a solution is presented. Finally representation mechanisms and programs necessary to implement E2EDE are presented at the end of the chapter.

Chapter 8 presents two case studies, one from the messaging system domain and the other from the database domain. In both studies, the problem specification is introduced first and then the details of applying the concepts of chapter 7. Finally it discusses and comments on the result of evaluation done by these experiments. More important the discussion focuses on the big picture of E2EDE applicability including how realistic it is to have standard PSM and what extended re-usability could be achieved and under what conditions.

Chapter 9 concludes the thesis by summarizing it and discussing the limitations and drawing out some points for future research.