A REQUIREMENTS SPECIFICATION MODEL FOR A PRODUCT LINE


2000 1289

A REQUIREMENTS SPECIFICATION MODEL FOR A PRODUCT LINE

SALAH KEMILEMBE KABANDA

(20044965)

2005

A REQUIREMENTS SPECIFICATION MODEL FOR A PRODUCT LINE

SALAH KEMILEMBE KABANDA

(20044965)

A dissertation submitted to the Faculty of Science and Agriculture in fulfillment of the requirements for the degree

of

MASTERS OF SCIENCE

in

COMPUTER SCIENCE

Department of Computer Science

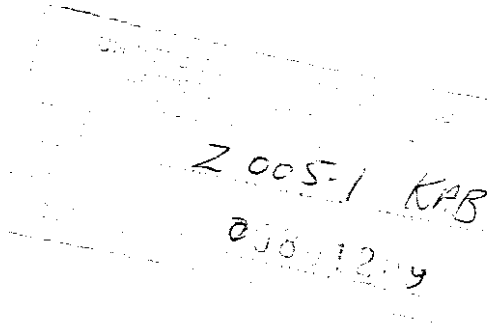
University of Zululand

December 2005

DECLARATION

This dissertation represents research work carried out by the author and has not been submitted in any form to another University for a degree. All sources used have been duly acknowledged in the text.

Salah Kabanda Date



DEDICATION

I dedicate this work to my mother, Leilah Rugachwa Kabanda, for believing and encouraging me through the difficult times and to my late grandfather, Tibangayuka Kabanda for instilling the value of education in our family.

ACKNOWLEDGMENTS

First, I thank the Lord for all the blessings bestowed on me in seeing this research work through. Secondly, I would like to extend special thanks to my supervisor Prof. M.O Adigun, who is most responsible for helping me complete the writing of this dissertation as well as the challenging research that lies behind it. I would like to thank his continuous fatherly support and mostly for believing in me when the challenge became difficult.

Besides my adviser, I thank all staff members and fellow colleagues in the Department of Computer Science at the University of Zululand for their continuous encouragement and support. Many thanks to Dr. Eyono-Obono of Durban Institute of Technology for having confidence in me and for listening to my complaints and frustrations. I thank Mr. P. Mudali for his support and assistance during the programming phase. Lastly, I thank my family who have put up with my busy schedule, but still provided me with their unconditional support, love and encouragement to pursue my interests. Your everyday prayers were not wasted.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATIONS

ACRONYMS AND ABBREVIATIONS

RE	Requirement Engineering
SDLC	Software Development Life Cycle
RSPL	Requirements Specification Model for a Product Line
RSL	Requirements Specification Language
RSD	Requirement Specification Document
MDA	Model Driven Architecture
WAF	Web tier Application Framework
PL	Product Line
PLSE	Product Line Software Engineering
DK	Domain Knowledge
KB	Knowledge Base
UP	User Perspective
MVC	Model-View-Controller
PLUC	Product Line Use Case

LOC	Lines of Code
TT	Transaction Throughput
W	Workload
TV	Transaction Volume
SE	Service Element
NFR	Non Functional Requirement
FODA	Feature-Oriented Domain Analysis
FORM	Feature-Oriented Reuse Method
SME	Small and Medium Enterprises

ABSTRACT

This research work focuses on developing a new requirement engineering model (RSPL) based on a Model Driven Architecture and Web-tier application framework, to support automatic and interactive requirements generation and specification when creating families of systems. In realizing the model, two goals were targeted namely (i) to construct a requirement engineering model that support automatic transformation of domain features into actor-specific requirements; and (ii) to design and implement an interactive web based requirement engineering tool that demonstrates the requirements generation and specification process for a product line. The result obtained is twofold: (i) a model driven architecture for rapid requirements generation and specification for a product line that reduces costs and development time; (ii) tool implementation based on a web tier application framework that supports different client and actor types. In conclusion, the study is a contribution to a recently advocated idea that requirements generation and specification for product line development could be model-driven. The result shows that the idea is promising with respect to requirement reuse and improving communication barriers among members of a system development team.

CHAPTER ONE

1.0 INTRODUCTION

1.1 Overview

Requirements Engineering (RE) is a set of activities that help develop the understanding of a system's domain, constraints, characteristics and systems functionalities as per stakeholders view, as well as the documentation of the systems specification for all stakeholders involved in systems development [7]. The deliverable from RE is a Software Requirements Specification document (RSD) that fully describes the external behavior of the application, its nonfunctional requirements, design constraints and other factors necessary to provide a complete and comprehensive description of the requirements for the software.

In the past, requirements engineers elicited and specified requirements using manual methods, which proved to be too slow and error prone to maintain a competitive position in the global economy. Therefore, the need for faster, better and less expensive methods of developing complex applications to meet market demands and still maintain a competitive status quo became desirable. The notion of automatic requirements tools was believed to be the solution to most challenges imposed by traditional methods of RE. Automatic tools implied that requirements are stored in a

requirements repository as individual objects instead of a paper document for easier maintenance and reusability purposes [12]. Although this was a breakthrough in the RE field, the challenge had just begun. Most tools were made for a single product development, making the task difficult when a family of systems had to be considered.

A family of systems is a set of applications with very similar requirements and few key differences that can be configured to provide reusable assets [24, 25, 26]. Developing software for a family of systems is not an easy task as in a single product development. It requires that the family domain be critically analyzed to identify and define common and variable features that can be used to create individual product instances.

The common features if exploited could achieve substantial cost savings (improvements in productivity; time to market; product quality and customer satisfaction) through reuse. This means that each time a new product is introduced to the domain the same asset repository can be reused leading to direct savings.

1.2 The need for new Requirement Engineering models

Tools and methods that assist with the production of better requirements have been reported in the literature [15, 16, 20, 32, 34, 35, 41]. Although they have been adopted in industry arena, poor requirements that do not meet stakeholders standards are still being produced, especially in the case of product line development. Poor requirements could be a result of:

i. tools inability to perform full RE tasks that meet stakeholder expectations;

ii. requirements engineers' inability to properly use the tool to its most potential due to the tool's lack of an interactive interface and its availability (web-based as opposed to traditional stand alone application) and

iii. lack of necessary features such as automatic requirement metadata generation and support for different client types (WAP and i-Mode clients).

Recently, Software Engineers have started to rethink RE models that are used to develop automatic tools. This work is a contribution to the idea that the Model Driven Architecture (MDA) and the web-tier application framework (WAF) could serve as a pattern for addressing RE problems. We are of the view that using MDA during the early stage of the software development life cycle, could positively impact and improve the requirements generation process.

1.2.1 Model Driven Architecture (MDA)

MDA is a framework that promises many benefits such as reusability, automatic code generation, portability, and many more. MDA is generally used from system design to implementation whereby a platform-independent model is automatically transformed into a platform specific

model through mapping principles. In other words, MDA converts a business model into a specific technology context (code) [3, 9].

Although MDA has worked very well during system development, it has not been specifically applied in the requirements engineering phase of system development. MDA features during the design and implementation phase of SDLC as it is concerned with transforming the design model (design phase) into a platform specific technology, i.e. code (implementation phase). The outcome is vast cost savings due to rapid system development and less errors due to lack of human intervention. This work is an attempt to extend the MDA benefits to the requirements phase. To realize even greater benefits, MDA will be supported with an interactive environment that supports multiple client types. The next sub section presents a discussion on a web tier application for a RE model.

1.2.2 Web tier Application Framework (WAF)

Most RE tools are still stand-alone applications that are location and time-dependent still have difficulty in supporting e-business application requirements and still have no support for various client types. A web tier application framework based on a Model-View Controller (MVC) pattern is one step forward in solving these problems. WAF provides interactive applications and provides a host of design benefits such as separating design concerns (data persistence and behavior, presentation, and control), decreasing code duplication, centralizing control, and making the

application more easily modifiable [13]. This work aims to use WAF as a supplementary framework to MDA in RE implementation.

1.3 Problem Statement

Advances in requirements engineering models and tools has still not produced high quality requirements that meet stakeholders' needs [41]. Most RE tools for a family of products do not produce requirement specification, but lay much emphasis on the elicitation, scoping and analysis phases.

Requirement Specification is an important component of the RE deliverable that helps to capture different views of various stakeholders. Each stakeholder has different views of the same system and wants their views to be considered. Conflicting views and goals require that stakeholders negotiate and arrive at realistic requirements that satisfy all parties. Negotiation becomes difficult if RE tools:

i. are stand alone application that are not interactive and difficult to access at any time and at any location and

ii. use a requirement specification languages (RSL) that stakeholders have to learn.

Web-based tools distributed and multi-user capability offer considerable potential to requirements engineers and developers. Important stakeholders who are often difficult to involve in the requirements process

can now contribute their goals, expectations, negotiation limits and preferences to the wider RE process [39] as time and location is no longer a barrier. This ensures that stakeholders specifications and expectations are properly integrated during system development, thereby producing 'better' requirements.

The Model Driven Architecture and a Web Tier Application framework have been identified as the architectural basis of this work. Using both MDA and WAF as starting points, the research questions to be answered are:

i. Can MDA's success in code generation be replicated in Requirements Engineering tasks (elicitation, verification and specification of requirements) for a family of systems?

ii. Can we, therefore, provide a reusable implementation of the model for different client types?

1.4 Justification

Much research in RE for a family of systems has focused on its initial stage – requirements elicitation and scoping, whereby the domain is defined to identify areas of potential reuse. Application development depends not only on elicitation and scoping of requirements, but greatly on requirements specification which describe what stakeholders want.

To produce better requirements while at the same time experiencing significant cost savings in terms of overhead costs and time, tools and RE models need to support the full RE phase, support requirements reusability and web-based. This research proposes to adopt the Model Driven Architecture as a pattern in addressing the mentioned aspects. The reason for adopting MDA in this work is because MDA:

i. Supports reuse when creating families of systems

ii. Encourages efficient use of system models in the software development process;

iii. Tends to address enterprise architectures supported by automated tools and services and

iv. Provides a conceptual framework for using models and applying transformations between them as part of a controlled, efficient software development process.

This work envisions an interactive requirements generation and specification model for a family of systems from which specific product line requirements can be automatically constructed from pre-existing domain knowledge. To ensure that requirements are automatically generated and specified, an MDA approach has been suggested. A web-tier application framework is adopted to ensure a web-based tool that is robust, scalable and easier to maintain.

1.5 Goal and Objectives

The goal of this research is to develop a requirements generation and specification model for a product line (RSPL) from pre-existing generic domain knowledge that has been analyzed, processed and structured.

The goal is decomposed into two specific research objectives:

i. To formulate a requirement engineering model for a family of systems that supports automatic transformation of domain features into actor-specific requirements and

ii. To develop an interactive web based requirement engineering tool that implements the model.

1.6 Research Methodology

In order to achieve each mentioned objective, the research uses the Model Driven Architecture approach and Web tier architecture to formulate the model for a family of systems that supports automatic transformation of domain features into actor-specific requirements. In addition, the study adopts a survey research design in which simple random sampling is employed. A questionnaire is used to collect empirical data from Small and Medium Enterprises (SME). This data is used to establish SME requirements for developing an interactive web based requirements engineering tool that implements the model.

1.7 Scope and Limitation

This work envisions an interactive requirements generation and specification model for a family of systems from which specific product line requirements can be automatically constructed from pre-existing domain knowledge. The tool does not have a validation capability essential for checking requirements completeness and consistency and nor was it designed with specific security, scalability and performance quality of service in mind. For it to be used in a production environment, design criteria for the Quality of Services issues must be evaluated first. The model does not also provide validation capabilities. These aspects of the research are deferred to the future.

1.8 Overview of the rest of the dissertation

The remainder of this dissertation is organized as follows: Chapter two provides a discussion of work done in RE, supporting tools and background information on the development and influential factors of the RSPL model. These factors include the Product Line Software Engineering, the Model Driven Architecture, and the Requirement Specification Languages.

Chapter three presents a detailed description of the model-driven RSPL architecture as well as the components that lead to the RSPL development. Then, a web tier application framework is discussed in relation with RSPL. Chapter four provides a typical illustration of how the

presented model could become the heart of an interactive, web based tool. Chapter five demonstrates an implementation of the RSPL tool using a case study and further presents the result obtained. Chapter six concludes the dissertation. Limitations of the study and recommendation thereof are also presented. Finally, directions for future work are suggested.

CHAPTER TWO

2.0 BACKGROUND CONCEPTS AND LITERATURE REVIEW

2.1 Introduction

In chapter one, RSPL was presented as a tool for responding to some of the challenges of requirements engineering. In this chapter, the foundational principles behind RSPL are discussed. These principles included but are not limited to Product Line Software Engineering (PLSE) and the Model Driven Architecture (MDA). The objective of the chapter is, therefore, to show that RSPL builds on a number of emerging software engineering issues.

2.2 Requirements Engineering Concepts

Requirements Engineering (RE) is the branch of systems engineering concerned with the *needs* and *wishes* of software-intensive systems, the goals to be achieved in the software's environment, and assumptions about the environment [46, 47]. While needs represent application constraints which can be supported by tools, wishes describe application solutions and are difficult to achieve as they are always changing due to changing business environment.

2.2.1 Users' Wishes

Wishes are desired properties that users would like the application to have. Wishes are complex because they are environment dependent. To understand them, requirements engineers iteratively set short targets that they can achieve within few days for each wish. Once completed, they meet with the user to deliver some quanta of business value. Thomas and Hunt [47] observed that the short targets or the concept of iterative development provided a way of *controlling costs; mitigating risks; capturing and verifying requirements* according to the user's wishes.

Iterative development led to agile methodologies such as extreme programming, SCRUM and Crystal methods among others. Agile methodologies attempt to minimize risk by developing software in short "timeboxes", called iterations, which typically last one to four weeks. Iteration includes all the tasks (planning, requirements analysis, design, coding, testing, and documentation) necessary to release a sub project [51].

Prior to agile methods, engineering methods tried to plan out a large part of the software process in great detail for a long span of time. This only worked well when there are no requirements change as result of business changes. Traditional engineering methods have a nature of resisting change and as a result deliver a product that does not satisfy user requirements [50]. Agile methods are adaptive rather than predictive

meaning that they welcome change and emphasize real-time communication, preferably face-to-face to enable easier incorporation of all users' who have to contribute their needs to the wider RE process.

2.2.2 Stakeholder Needs

On the other hand, stakeholder needs are supported by tools that facilitate the requirements engineering process. Software tools that support software engineering tasks are typically available as stand-alone applications and their advantages have been practically observed by industries and Software Development Team [40]. Software tools increase productivity, decrease overhead costs and due to limited human intervention, ensure that qualitative products are developed.

Most RE tools are elicitation, communication, modeling, verification or management oriented tools or a combination of one or two of these features and have been designed or developed for a single product instance development and not for a given domain or a family of systems.

The term domain is used to denote a set of systems or functional areas, within systems, that exhibit similar functionality [21]. Domain engineering is the foundation for emerging "product line software engineering" approaches that affect the maintainability, understandability, usability, and reusability characteristics of a system or family of similar systems.

2.3 Product Line Software Engineering

The body of knowledge known as automatic requirements engineering has received considerable attention from researchers as can be seen in literature [10, 15, 16, 24, 25, 33, 34, 35, 38, 41]. This aspect of software engineering developed from the concept of family of systems was first suggested by Parnas in 1976 [35]. Parnas was of the opinion that substantial savings could be achieved by reusing the common features in programs that are developed as a family. The widespread attention given by the research community resulted in Product Line Software Engineering (PLSE).

PLSE is an emerging software development alternative to developing every software system from scratch. However, PLSE will only be adopted if it makes business sense, that is, helps shareholders maximize their profit. Whether this investment results in greater profit depends on the particular strategy adopted.

To this end, a number of methodologies have been proposed to address this challenge. These methodologies require that a domain model be constructed to document all domain artifacts. The model results from domain analysis whose goal is to scope the domain with an aim of identifying similarities and variations. Once the domain model is constructed, it is documented in the Domain Reuse Library as a reusable artifact for application development.

Typically as shown in figure 1, a domain reuse library is built for creating reusable requirements specification from raw domain requirements and secondly cataloging the specification as reusable artifacts in the library. It is a repository for storing reusable assets and allowing users to search for assets in the repository [49].

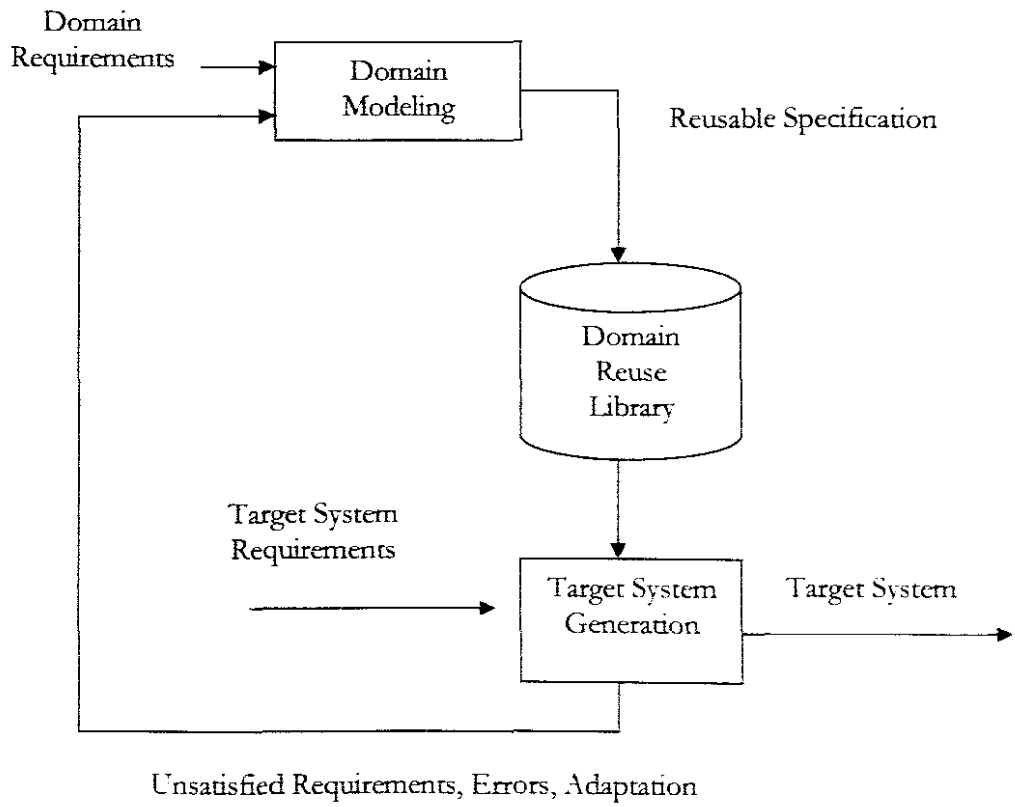


Figure 1: Domain modeling for a family of systems [16]

The following sub section describes a number of proposed Product Line Software Engineering methodologies for requirements engineering.

2.3.1 Product Line Software Engineering Methodologies

2.3.1.1 Kernel and View approach

The Kernel and View approach was introduced by Gomaa [15] who was of the view that a domain model should constitute the core of PLSE as it represents the entire family requirements. According to Gomaa, the domain model is formulated from one of the either two approaches namely the kernel *or* view approach.

While the view is an integration of multiple viewpoints (the Aggregation Hierarchy, Object Communication Diagrams, Generalization/Specialization Hierarchy, Feature /Object dependencies and the State Transition Diagram), the kernel encapsulates the common features that represents one or more domain requirements [6, 10, 17, 26]. Individual members of the family are then generated by tailoring the domain model constructed from either a kernel approach or a view approach. This study requires that the domain model be viewed from both the kernel and the view perspective as both describe the domain differently. Kang et al suggested the FODA approach that allowed the integration of both perspectives during product line development.

2.3.1.2 Feature –Oriented Domain Analysis (FODA)

As both the kernel and the view approach intend to describe the domain, Weiss and Kang [24, 25] suggested that *both* methods should be considered/ integrated in the requirements engineering process for a

family of systems because they compliment one another in the development of an application instance of the domain. They proposed a two-fold approach for a family of systems (i) domain engineering and (ii) application engineering. During domain engineering, domain experts strive to identify and define product family requirements in terms of *features* while during application engineering, family members are produced by selecting or tailoring features that will constitute the product instance.

The use of features is motivated by the fact that customers and engineers often speak of product characteristics in terms of features the product has and/delivers [48]. The use of features led to the Feature–Oriented Domain Analysis (FODA) approach that defines techniques for developing, parametrizing and configuring reusable assets plus a specific process of commonality and variability analysis [26]. The result is a feature model that supports both the domain engineering of reusable artifacts and development of applications using domain artifacts.

Although stakeholder requirements (features) are essential inputs for core asset development, they are not sufficient on their own as they are constantly changing with the business environment. This study requires that the business environment should also be incorporated into the domain model together with the stakeholder requirements. Kang [24] suggests that FODA be extended to support a marketing and product plan

perspective (MPP) as well as explore analysis and design issues from that perspective. MPP helps to identify the information to gather during the marketing and business analysis. Such information includes marketing analysis and strategies, product features and delivery methods. These suggestions led to the development of the Feature-Oriented Reuse Method (FORM).

2.3.1.3 Feature-Oriented Reuse Method (FORM)

FORM is a systematic method that focuses on capturing commonalities and differences in a domain in terms of *features* and using the analysis results to develop domain architectures and components. FORM consists of two major processes namely (i) asset development – a process that includes capturing and analyzing the commonalities and variabilities such as marketing and product plan development and refinement; feature modeling and requirements analysis and a (ii) Product development – a process that entails analyzing requirements and selecting and adopting features for the product [24].

FORM starts with capturing domain information and analyzing common elements in terms of services, operating environment, domain technologies and implementation techniques to generate a feature model. A feature model is used to define *parameterized reference architecture and appropriate reusable components instantiatable during product development* [48]. During product development, product features are

selected from the domain/feature model to be instantiated. Product features are classified into two main categories: Functional and Nonfunctional. While Functional Features (Functional Requirement - FR) are services that provide behavioral characteristics, which define the activities inherent in the domain; Nonfunctional Features (Nonfunctional Requirements - NFR) are end-user-visible application characteristics that cannot be identified in terms of services or operations [24]. They present *the qualitative aspects of the domain.*

Moon et al [32] suggest that product features for a family of systems be represented using a Requirements Metamodel. A metamodel is a model *that describes another model. It consists of appropriate constructs* reflecting the declarations of data-definition and data-manipulation languages that represent the constructs or the building blocks of the product. The metamodel also provides a bridge between organizational structure aspects and business subprocess, minimizing the complexity of business process definition and at the same time improving the efficiency and quality of it [32, 37]. Its purpose, according to Moon et al [32] is to *lay down an overall scheme for representing domain requirements.*

Figure 2 presents a typical metamodel for domain requirements. The core model element of a requirement metamodel is the *Domain Requirement* as it represents one general requirement that can be reused as a core asset of developing systems in a product line [32]. The Domain

Requirement consists of Functional Requirement and Non Functional Requirement with variants to distinguish one product feature from the other. Variations are captured using the Variability element and can either be optional or mandatory. A mandatory feature signifies commonality and potential area of reuse. Functional Requirement consists of services rendered to other parties within the domain (either to other domains, products within the domain or to another service). A service defines the functionalities or activities which a product within the domain pursue to fulfill their goals. Services are, therefore, like use cases or scenarios. A service can have multiple sub services and can also be refined to a service element.

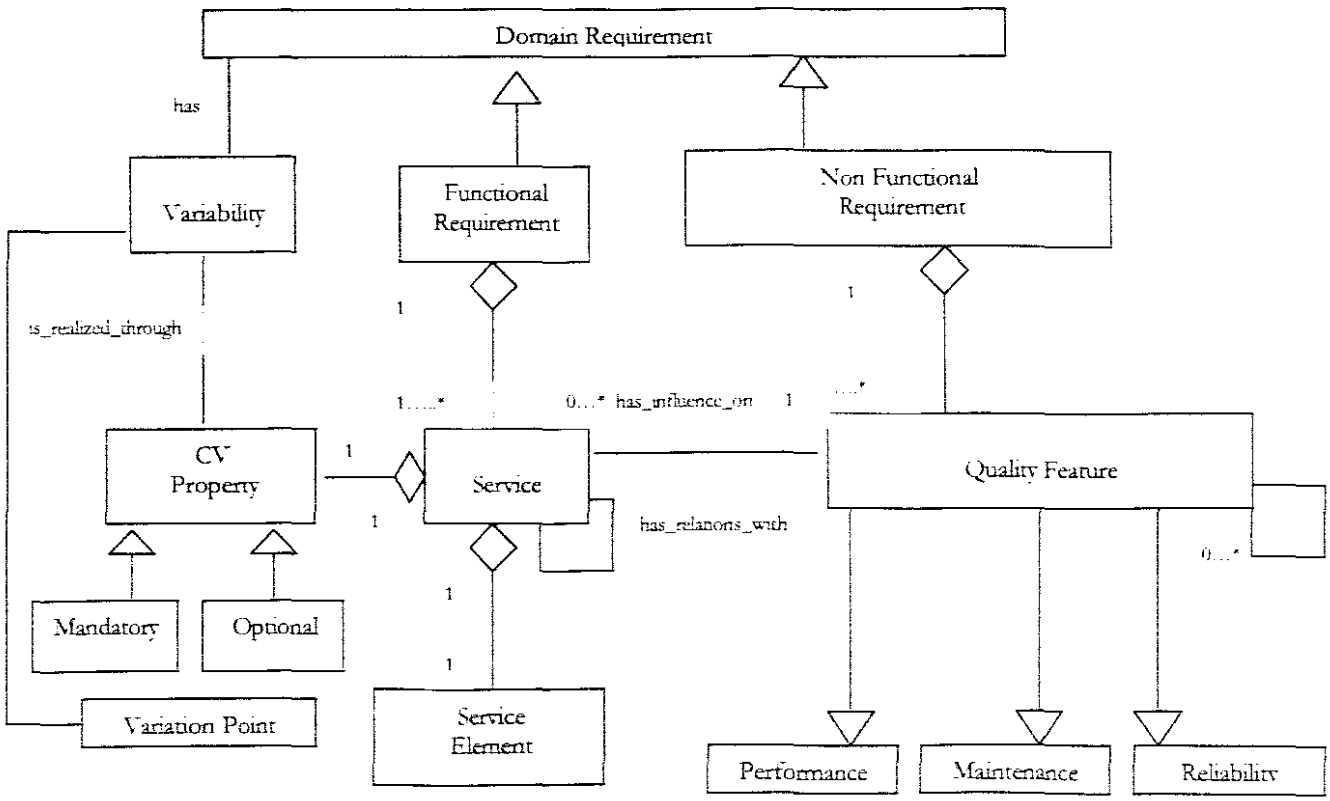


Figure 2: A Requirement metamodel for a product line [32]

A service element is the smallest unit that cannot be further decomposed to sub services. A service can be influenced by NFR features. NFR is composed of three qualitative features: performance, maintenance and reliability.

a. Performance

Performance is concerned with the efficiency rate at which the application performs its functionalities when subjected to a particular workload. It is affected by the available resources and how they are used and shared. It is during the requirements phase of the SDLC that performance objectives, workflow, and key service elements (scenarios) are defined. The workloads and estimated volumes for each service element are then considered [1]. There are two performance objectives that this research pays attention to: Transaction Throughput (TT) and Workload (W). TT is the number of requests (service elements) that can be served or completed by the application per unit time (T), for example, fetching and updating a row. We, therefore, measure TT as the number of service elements per time unit as depicted in equation 1.

$$TT = \frac{SE}{T} \qquad \text{..... Equation 1}$$

Workload is total number of users and concurrent active users (A) at a given time unit or transaction volumes (TV) expected to be handled by the

application in a given time unit (T) [1]. Equation 2 depicts a workload equation using transaction volumes.

$$TV = \frac{V}{T} \quad \text{.....Equation 2}$$

When actors are building or generating requirements for an application instance, the workload which their application will handle has to be estimated. The transaction volumes to be expected are determined by the number of features which a domain possesses.

b. Maintenance

Maintenance is concerned with the component's ability to change when requirements change. According to Bosch, change categories tend to be organized around the interfaces the application has to its environment and are best captured using change scenarios. Change scenarios are given a relative weight which indicates the likelihood of that scenario occurring during that period. Such change is measured by the number of lines of code (LOC) that have to be changed to accommodate the scenario or service element. An actor is, therefore, expected to provide an estimate of LOC that will be required to change over the life cycle.

For each feature, the cost for developing a service element (SE_{CHARGE}) is determined. Then a maximum range ($SE_{MAXCHARGE}$) is imposed on the existing LOC to determine the highest estimated cost of maintaining that

service element. When there is a change to be expected as a result of changing environmental requirements, the LOC of the service element to be changed should not exceed LOC of $SE_{MAXCHARGE}$ as indicated in equation 3.

Service Element Maintenance cost (minimal) = SE_{CHARGE}

Service Element Maintenance cost (maximum) = $SE_{MAXCHARGE}$

Cost of Service Element to be changed = newSE

$SE_{CHARGE} \leq newSE$ ..... Equation 3

c. Reliability

Kan [2, 23, 42] proposes that reliability can be measured using two metrics: defect density rate (DDM) and Mean Time To Failure (MTTF). DDM measures defects relative to software size measured in lines of code during a specific time frame.

$$DDM = \frac{\text{Defects}}{\text{LOC}}$$ ..... Equation 4

The expected number of defects over a certain time period is important for cost and resource estimates of the maintenance phase of the software life cycle and, therefore, is rendered as a beneficial metric for reliability and maintenance phase of the software life cycle [22, 23, 43]. Another benefit of DDM is that it can be applied to general-purpose computer systems or commercial-use-software for which there is no typical user profile of the

software. In a product line oriented approach, a generic assessment of the *domain (common reliability metrics) can be used as a building block for* reliability assessment of application instances in the domain.

MTTF measures the time between failures. Implementing MTTF for general-purpose computer systems or commercial-use software, for which there is no typical user profile of the software is more difficult than for special-purpose software systems such as the air traffic control systems or the space shuttle control systems. MTTF requires that the operations profile and scenarios/service elements should be defined, the activities and sequential occurrence in the software system should also be provided. For generic system, this would be time consuming and an expensive process to have to record the occurrence time of each software failure at a high degree of accuracy for the results to be useful.

Bosch [1] described the quality attributes in terms of profiles and associated properties. A profile is a set of scenarios, generally with some relative importance associated with each scenario [1]. The different types of profiles for each quality attribute are depicted in table 1.

Quality	Associated profile/scenario
Performance	Usage
Maintainability	Change
Reliability	Component interaction.

Table 1: Quality Profiles

The Performance quality is associated with the general efficiency with which the system performs its functionality and, therefore, key words that characterize it include: response time and/or usage scenarios. The Maintainability quality is associated with how the architecture can be affected by requirements change and therefore has change(s) scenario as its main key word. Reliability is associated with components interaction during operation and the effects of component error.

2.3.2 Product Line Software Engineering supporting tools.

As Product Line Software Engineering Methodologies continue to emerge, so is their adoption in the development of automatic requirements tools for a family of systems. A support tool is a natural consequence of industrial practice if it is to become mature. This explains why systems engineering and software development organizations are under pressure to construct tools that would automatically generate requirements and even an entire system code. A number of tools are, therefore, coming into existence as natural evolution from methodologies.

2.3.2.1 Elicitation Tools

Requirements elicitation is an early process of the RE that tries to capture the information and knowledge of the system under construction [12]. Figure 3 shows a typical requirements elicitation process, whereby information needed to build requirements specification for a single system or a product line model is normally elicited by domain expert with

knowledge in the problem or application domain (the processes and products in product line engineering).

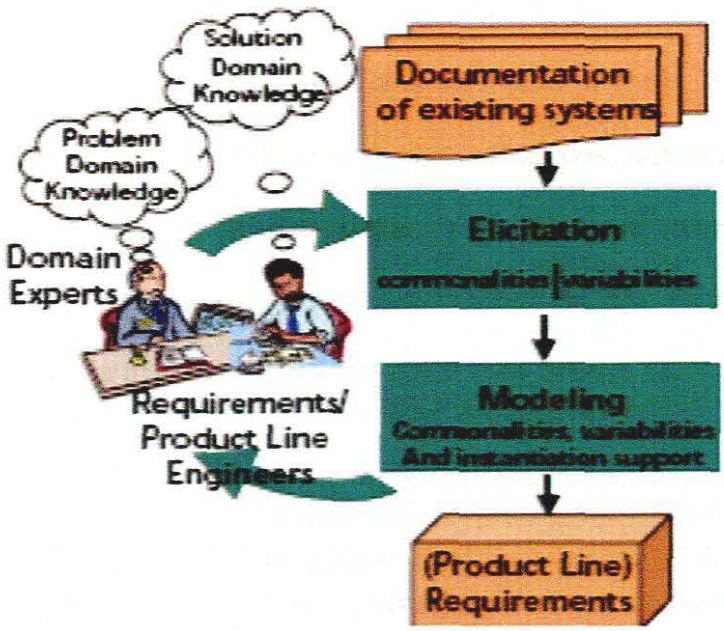


Figure 3: A requirements elicitation process

Use Cases have been proposed as an effective approach for PLSE elicitation to capture product features (especially functional requirements) for software systems [55, 56]. Use cases allow structuring requirements documents with use goals and provide a means to specify the interaction between an actor and its environment [5]. The term actor is used to describe the person or system that has a goal against the system under discussion. There are two main actors namely the primary and the secondary actor. While a primary actor triggers the system behavior in order to achieve a certain goal, a secondary actor interacts with the system but does not trigger.

According to Fantechi et al [56], a Use Case is completed successfully when its goal is satisfied and is extensively described in the Use Case "Description". Use Case descriptions also include possible extensions as reflected in Cockburn's Use Case in Table 2 [56]. The "Description" section is express in natural language sentences, describing a sequence of actions of the system while Variations are expressed (in the "Extensions" section) as alternatives to the main flow, linked by their index to the point of the main flow from which they branch as a variation [5].

USE CASE#	<the name is the goal as a short active verb phrase>	
Goal in Context	<a longer statement of the goal in context if needed>	
Scope & Level	<what system is being considered black box under design> <one of: Summary; Primary Task, Sub Function>	
Preconditions	<what we expect is already the state of the world>	
Success End Condition	<the state of the world upon successful completion>	
Failed End Condition	<the state of the world if goal is abandoned>	
Primary, Secondary Actors	<a role name or description for the primary actor> <Other systems relied upon to accomplish the use case>	
Trigger	<the action upon the system that starts the Use Case>	
Description	Step	Action
	1	<put here the steps of the scenario from trigger to goal delivery, and any clean up>
	2	<....>
	3	
Extension	Step	Branching Action
	1a	<condition causing branching> <action or name of sub-Use Case>
Sub Variations	Step	Branching Action
	1	<list of variations>

Table 2: Textual Use Case template [5]

Variations are described and specified by tags that indicate those parts of the product line requirements needing to be instantiated for a specific product in a product related document. The tags represent three kinds of variability as defined by Fantechi in [56]:

i. Alternative, *expressing the possibility to instantiate the requirement by selecting an instance among a predefined set of possible choices, each of them depending on the occurrence of a condition;*

ii. Parametric/Mandatory, *from which instantiation is connected to the actual value of a parameter in the requirements for the specific product; and*

iii. Optional *of which the instantiation can be done by selecting indifferently among a set of values, which are optional features for a derived product.*

An example of use Case in the Product Line Use Case (PLUC) notation, taken from [56] is presented in Figure 4. The Use Case describes the activities related to the submission of a project document. They suppose that it can be possible to submit different two types of documents: slides (in the .ppt format) or papers (in .doc, pdf. or .ps format), variables (here V1 and V2) describe the variation points within the use case.

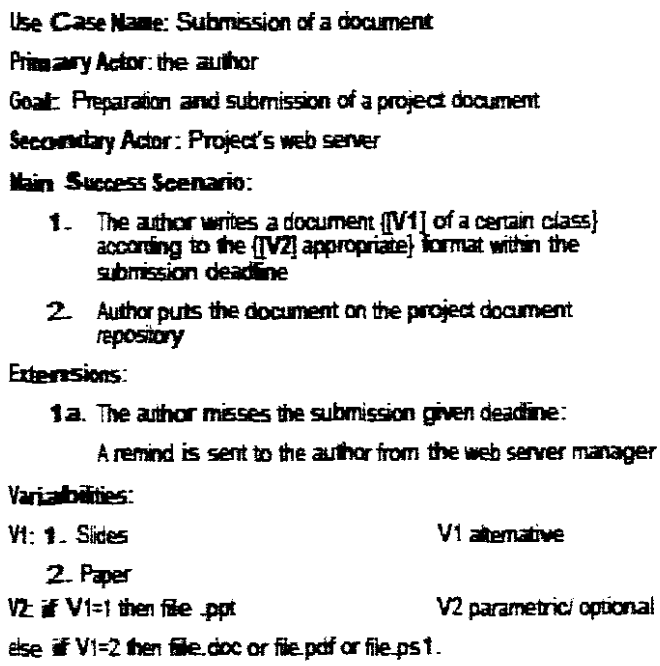


Figure 4: Example of a Use Case in the PLUC notation taken from [56]

The possible instantiations and the type of the variations is given within the use case and the possible values are described with logical expressions.

2.3.2.2 *Scoping Tools*

Domain scoping is an activity that helps determine the systems and the system features that should be included in the system family [57], because it may not be beneficial to include all systems and features in the system-family. Most scoping tools have elicitation functionalities as well. Examples of scoping tools include TruScope [8] and PuLSE BEAT [39].

2.3.2.3 *Verification Tools*

Before developing a particular feature or component of a product, it is important to verify that the system family supports all features defined

during scoping, that it can be easily instantiated for each of the systems in the system family and that planned new features can be easily incorporated [57]. Verification tools attempt to confirm the completeness and consistency of new members of the product line. An example of verification tool is DECIMAL [34].

2.3.3 Analysis of Product Line Software Engineering Tools

A thorough analysis of the product line software engineering tools reveals that:

i. Most of the tools do not support the full life cycle of requirements generation. They either focus on one or two of the phases of requirements engineering such as the elicitation, scoping or verification phases and

ii. Most tools are stand-alone applications that lead to unnecessary time wastage, hinder communication among stakeholders and reduce the return on investment.

Tools and methodologies which support the full requirements generation process while at the same time ensuring minimal development time and increased return on investments are still to be researched into. This research work presents the Model Driven Architecture approach to RE.

2.4 The Model Driven Architecture Approach (MDA)

The code generation expectation is not a recent development, what is new is the recent interest in Model Driven Architecture (MDA) as a model for code generation. MDA consists of three viewpoints namely, the Computation Independent Model (CIM), the Platform Independent Model (PIM) and the Platform Specific Model (PSM) [52].

The MDA starts with CIM which represents a model of the business that is completely independent of any IT systems (structure and implementation of the system are hidden, or possibly not yet implemented). The second viewpoint, the PIM focuses on the operation of the system while hiding the platform-dependent details, that is, an implementation-independent model of the functionality of an IT system. It may use a general-purpose, platform-independent modeling language such as UML. The final viewpoint, the PSM focuses on the implementation details of a certain platform. It is a technical model of an IT system that considers architectural constraints given by a chosen platform (e.g. J2EE, .NET or CORBA).

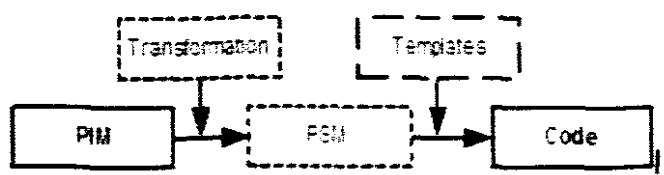


Figure 5: The flow of an MDA generator

The Model Driven Architecture approach aims at separating the business logic in PIM from the underlying platform specific module/technology and

represents this logic with precise semantic models as depicted in figure 5. The MDA assumes that the design already exists and represented as CIM. Given a requirements design in UML (Unified Modeling Language), an MDA generator transforms the requirements into its equivalent code based on its specific platform technology. The MDA is becoming popular due to a number of reported benefits [3, 9, 43] such as:

i. reduced cost throughout the application life-cycle;

ii. reduced development time for new applications;

iii. increased return on technology investments (ROI) and

iv. rapid inclusion of emerging technology benefits into existing systems.

The MDA is not a RE generation and specification approach but an approach for automatically generating code from a design, resulting in quick system development and return on investment.

2.5 RSPL as a tool for PL Requirements Generation and Specification

Our approach to requirements generation and specification for product lines is based on the Model Driven Architecture and supported by Kang's Feature Oriented Reuse Method (FORM) to scope and elicit product line features. The use of the Model Driven Architecture is based on the notion that requirements can be automatically generated from pre-existing

domain information that has been analyzed, processed and structured to suit a particular standard Requirements Specification Language. RSPL is a contribution to RE with the following characteristics:

i. a scoping and elicitation tool for a family of systems;

ii. a specification tool for describing the requirements for a family of systems; and

iii. an enabler for bridging the communication gap between the software development team members, with particular reference to software engineers and domain experts.

The RSPL approach, therefore, allows the separation of the interface logic from the underlying business logic. The interface logic is the point of interaction between users (domain expert, analyst and developer) and the tool. The domain Expert's role is to define the domain boundaries captured as domain features characteristics. The process involves domain definition, characterization and scoping. The analyst's responsibility is to identify and define individual product instances. A product instance is defined by its business functionalities and domain-specific behaviors. The developer's position and relationships with RSPL is to refine the requirements into design-usable form. This is achieved by adding perspectives that transforms domain requirements into "ready-to-use" requirements specifications.

The RSPL process is actor-driven, therefore, a portal-type interface to the tool is proposed. Each user is associated with an actor use case template that drives an actor-specific interface of the portal. In the heart of this portal is the RSPL tool which transforms each user's contributions into domain knowledge. The tool uses appropriate rules to transform the knowledge into Requirements Specifications that are ready for a software engineer to use.

CHAPTER THREE

3.0 MODEL DEVELOPMENT

3.1 Overview

A typical requirements engineering process involves all stakeholders (customer, domain expert, analyst and developer), but it is mainly the job of the requirements engineers to generate specifications from specified needs. While there has been previous work in the literature that attempted to develop automatic requirements tools for a family of systems [1, 2, 29, 38], most are either scoping or modeling tools. They do not generate specific requirements nor do they offer support for all client types, thus leading to communication barriers among the software development team members (Domain Expert, System Analyst and Developer) and other stakeholders such as the customer. Existing tools are mainly designed with the System Analyst's knowledge in mind with little consideration for the developer's perspective. Hence, the need to delineate roles in the software development process.

We associate requirements engineers with the actual initiation of the application development process which entails: eliciting domain information, identifying domain features, and developing application instances of the domain. There are three actors playing the

requirements engineering role namely Domain Expert, System Analyst and the Developer.

The reasoning behind delegating requirements engineering to three actors is that:

i. Domain experts are not expected to be familiar with the software development process but are a good source for domain knowledge information [35] and

ii. System Analyst and the Developer are responsible members of the software development team who define requirements for an application instance. But what a System Analyst considers input to a given model or component, the developer could regard as an output element. This phenomenon which could constitute communication barrier is totally avoided in our scheme.

The RSPL model automates the interactive construction of generic product line requirements. The model consists of a tool for producing specific textual application requirements. RSPL aims at bridging the communication gap between the software development actors, thus enhancing the requirements generation process. The tool is GUI (graphical user interface) driven to enhance easier actor interaction and has automated functionality for the different levels of actors.

The Domain Expert's role is to identify domain characteristics that define and distinguish one domain from another. The role entails interacting with the customers to identify user requirements. The knowledge gathered during the interaction is stored in a Domain Knowledge base for the analyst and developers to utilize.

The System Analyst role is to analyze the content of the domain knowledge in order to identify product instances and their features. During this process, all products that belong to a domain are identified and their features defined. Features abstract a products characteristics, functionality and relational constraints. Features provide system analysts with a clear distinction of potential areas where reuse can be applied. The Developer's role is that of tailoring domain knowledge to suit his preferences whilst generating requirements. He chooses specific requirements from a set of existing generic requirements.

3.2 A Model-Driven Requirement Specification for a Product Line

RSPL posits that, given a domain knowledge, it should be possible to generate specific application requirements using various transformational rules and templates. First, RSPL requires the user to elicit, scope and store domain knowledge in the knowledge base. The domain knowledge is generic in nature in the sense that it does not reference any particular system implementation or technology. In other

words, it is based on a computation-independent model (CIM) that captures the information viewpoint of a target application or a conceptualization of the application.

Second, actors using their expertise and experience, tailor domain knowledge to suit requirements for a specific application instance. Tailoring of requirements entails transforming the generic domain knowledge into specific requirements. Finally, requirements are then generated using standardized templates. Therefore, RSPL adopted the Model Driven Architecture as a pattern for generating prototype requirements from domain knowledge.

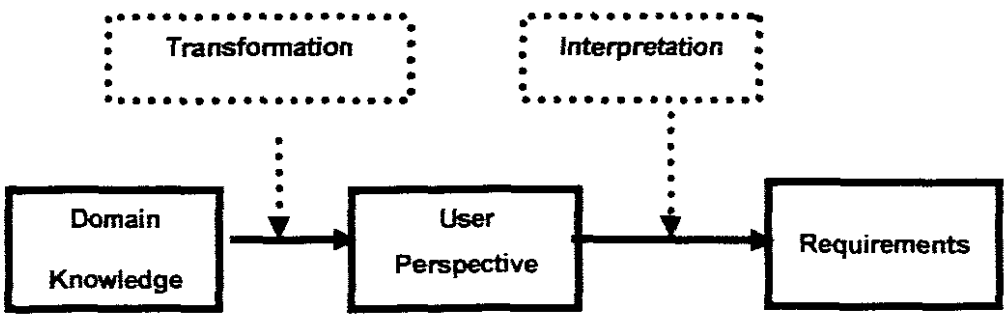


Figure 6: The Model Driven RSPL Architecture

RSPL defines three main components shown in figure 6, namely: Domain Knowledge, User Perspective and Requirements component. The first component Domain Knowledge stores all elicited and documented requirements. These requirements are generic in nature because they represent domain features. Secondly, we have the User

Perspective component, an actor-oriented mechanism for tailoring the domain knowledge according to the actor's preferences to suit the product whose requirements is to be generated. The third component is generated automatically by interpreting actor-specific knowledge into requirements specifications.

3.2.1 Domain Knowledge

The Domain Knowledge (DK) is a central repository that supports documenting and processing of domain knowledge artifacts. Two main processes are associated with DK as a knowledge base:

i. *Elicitation and scoping* of domain features, which involve capturing of domain information and identification of a domain scope i.e. the task of bounding the domains that are supposed to be relevant to the product line [8]. It is a process of gathering information; recognizing reusable assets and core artifacts of the product line to form RSPL requirement metamodel. RSPL Requirement metamodel is presented in figure 7. It is based on the Requirement metamodel of figure 2. The RSPL metamodel emanates from three interrelated objects namely Domain, Product and Feature. Each object is explained in detail under the User Perspective component in section 3.2.2.

ii. *Evaluation and documentation of elicited features.* This process involves analyzing the impact of each identified asset in terms of the risks it poses and gains it will provide to the product line; and documenting the artifacts as relational objects.

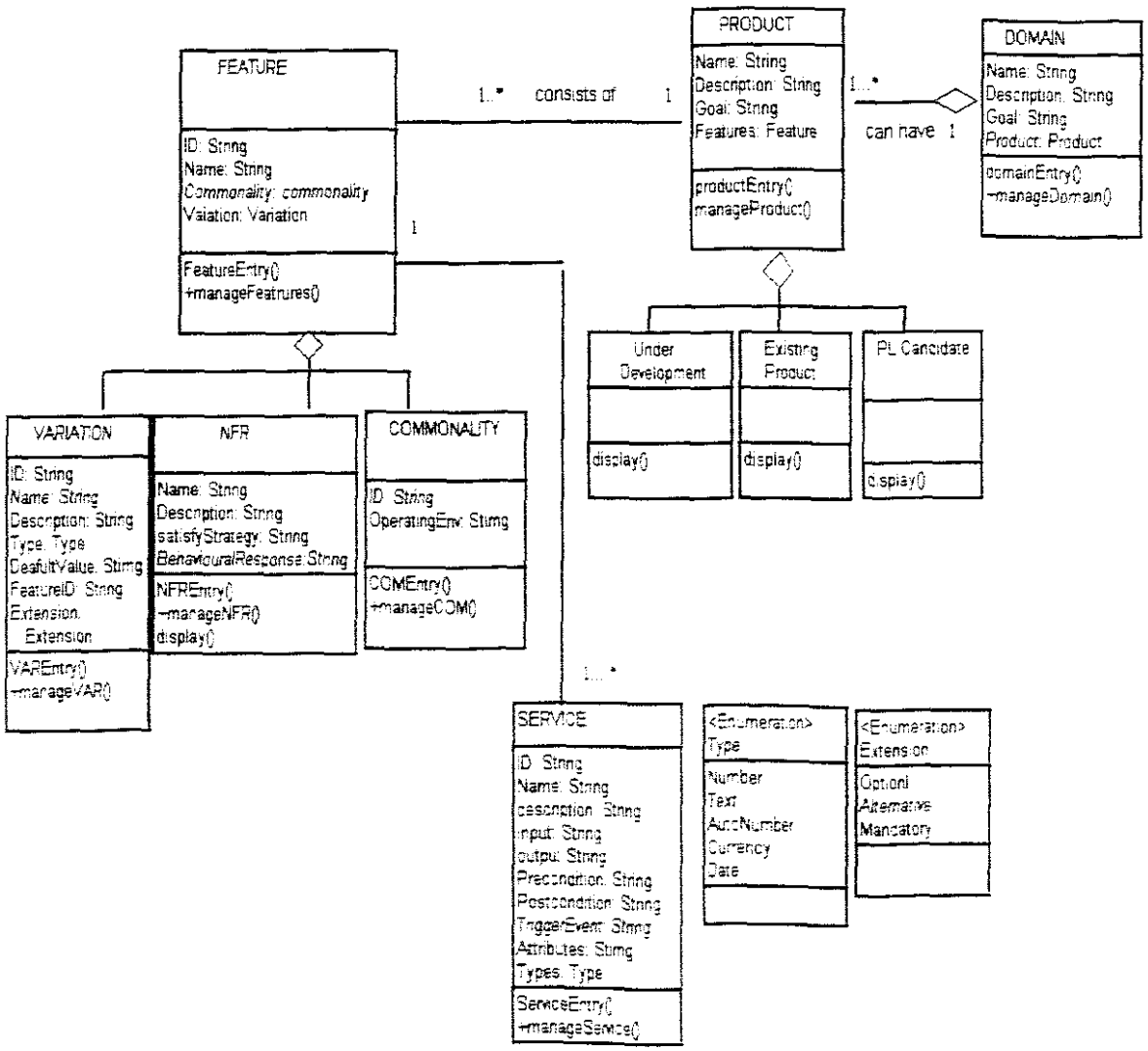


Figure 7: Product Line Requirement Metamodel

3.2.2 User Perspective

The User Perspective (UP) component is a portal-type interface in which the requirements elicitation, scoping and specification take place with the aim of gathering the expert knowledge of the application domain and specify requirements. UP implements the tailoring of the DK information to suit the application to be developed. Tailoring activities include requesting, manipulating and accepting user-type data pertaining to the problem domain.

There are two main templates designed for elicitation and scoping of domain information: Domain Template and Product Template. The templates originate from the Domain and Product objects in the RSPL metamodel (see figure 7). The templates define features of the domain or product in terms of domain-specific or product specific services. The techniques used in the template include questionnaire elicitation approach that consists of both closed and open-ended questions. This technique enables sufficient information to be captured and allows flexibility. Close-ended questions are more specific and take up less time than open-ended questions. The questionnaires are set up as *Textual Use Cases* as suggested by Cockburn [39] and Fantechi et al [55] to elicit domain and product information.

3.2.2.1 *Domain Object Template*

A domain template provides the mechanism to collect information pertaining to domain characteristics and constraint relationships with other domains. The essence of capturing domain characteristics is to specify information that uniquely differentiates one domain from the other. Figure 8 is an illustration of the RSPL domain template.

Naturally, domain information includes the domain name, followed by a short but comprehensive narration of what the domain entails and its expected features. The other elements are goal and objectives the domain is pursuing; an actor who is either primary or secondary depending on the role played; variation points expressed by using the term *Extension*; and variation status (optional, alternative or mandatory) captured as the *Variabilities* feature.

Domain Name:	Name that uniquely identifies the domain
Description:	Short but comprehensive narration of what the domain entails and its expected features
Goal:	A longer statement of the goal in context if needed
Actor:	Main user of the domain
Other Actor:	Secondary users
Extensions:	Variation points
Variabilities:	Variation Status

Figure 8: Domain Object Template

3.2.2.2 *Product Object Template*

A product template elicits the features of a product in terms of its functionality and feature relationships. The template is derived from defined product features. An RSPL product template consists of elements such as:

i. **A list of product characteristics:** These include product identity (ID); product name, product type, goal, actors to name a few (see figure 9.0). The product type determines the category of the product. A product can be in any one of the three main categories: (i) a product already in existence within the family or domain; (ii) a product that could become an instance of the product line, implying that it does not exist in the domain but is a viable product line candidate; or (iii) a product that has been identified for future instantiation in the product line, that is, the product has been acknowledged in the domain but has not yet undergone development.

ii. **A set of functional requirements**: The behavioral characteristics of a product are elicited by using the Service property on the template. Each service property is itself described on a Functional Use Case template (see figure 9.1). Associated with a service are other features such as event trigger, which activates the use case; and pre and post

conditions which provide the state of the world before and after the use case is executed.

iii. **A number of variation points:** Variability in a product is expressed as the presence, absence or substitute of Service features. Therefore, each product instance has variation points called extensions. An extension is a named service feature: optional, alternative, or mandatory. An optional feature may be present or absent. An alternative connotes the existence of a substitute and mandatory implies the features must be present in the product instance.

iv. **A set of quality constraints:** The Product template has a Constraint feature, used for specifying possible constriction on the product. Typical constraints include quality attributes such as performance, reliability and maintenance characteristics of the product. Each quality attribute is specified by using a *Nonfunctional Use Case Child* Template. The template captures quality characteristics using features called (i) Notion and (ii) Behavioral Responses features (figure 9.2). Notion is a brief description of a product's NFR. The behavioral response specifies the implication of each use case in the Product using a set of predetermined profile scenarios from Bosch [1] such as

usage scenario for Performance and change scenario for maintenance requirements.

v. **Relational Constraints:** If we assume that the domain is a universe, then products constitute sets in the universe. Normal set relationships are defined between products, the features from which products are composed and the services that interact to implement a feature. A formal description of the relational constraints is outside the scope of this dissertation.

Product ID:	An identifier to uniquely identify the product from others in the domain
Product Name:	Name of the product instance
Description:	Short but comprehensive narration of what the product entails and its expected features
Goal:	A longer statement of the goal in context if needed
Product Type:	Product category or differentiation
Actor:	Main User of the domain
Other Actor:	Secondary users
Services:	Use case states describing product functionality
Extensions:	Express variability
Variabilities:	Variability types and constraints

Figure 9: Textual Product Object Template

Use Case ID:	An identifier to uniquely identify the use case from others
Use Case Name:	Name of the use case
Goal:	A longer statement of the goal in context if needed
Actor:	Main User of the use case
Other Actor:	Secondary users
Preconditions:	What we expect is already the state of the world
Post-conditions:	State of the world upon successful completion
Trigger:	Action upon the system that starts the use case
Extensions:	Express variability
Variabilities:	Variability types and constraints

Figure 9.1 *Functional Use Case Child Template*

Use Case ID:	An identifier to uniquely identify the use case from others
Use Case Name:	Name of the use case
Notion:	Elicit meaning of the use case and its fundamental relations with other use cases.
Behavioral Response:	Specify the connotation of the use case in the product

Figure 9.2: Non Functional Use Case Child Template

3.2.3 The Requirements Generation Process

The Requirement Generation process is dictated by both the domain and product templates. When an actor creates a product instance by tailoring domain and product features, the *requirements generation process* is activated. The *requirements* generation process consists of the following steps:

i. *Selection of a template* in the domain: An actor triggers the requirements-generation process by selecting a domain from

which requirements are to be generated. Each domain is characterized by one or more generic product templates. RSPL presents these generic product templates to the actor who makes a selection of one of the templates.

ii. *Selection of one of the listed instances or creation of a new instance*: RSPL retrieves a number of existing product instances that are related to the specific pre-selected product template, from domain knowledge. The actor chooses the product instance that meets his current requirements or creates a new product instance if none of the available instances fits his current wishes, a new product instance is created by tailoring the domain knowledge and identifying variation points.

iii. *Retrieve the corresponding data set from the Domain Knowledge*: Once an instance is selected or created, RSPL retrieves the data set corresponding to that product instance and maps it to the RSD document features.

iv. *Populate Empty RSD with data set value:* Each time a data set is selected from the Domain Knowledge, the RSD as reflected in figure 10, is automatically populated with corresponding data set and becomes the generic reference for product instances within that category.

The deliverable from the for-step process above is a requirements specification document (RSD). An RSD is, therefore, a union of both specific domain and product template features that is automatically generated during the requirements generation process. Moreover, an RSD is made up of five parts namely (i) an Introduction or epilogue section which presents requirements overview of the application instance to be developed; (ii) Functional Requirements section, which outlines features to be reused for each implementation, without being re-invented in each separate implementation; (iii) Quality Requirements section that presents the NFR properties; (iv) a variability section, what and how services will vary; and (v) a Glossary section of domain terms. Before an actor invokes the requirements specification process, the RSD document parts are empty. Each of the parts is filled in once the requirements specification process is completed.

Figure 10: A Requirements Specification Document

3.3 A Web-Tier Application Framework for RSPL

A web-tier application framework design, based on the Model-View-Controller (MVC) pattern has been adopted in this research with a view to make the tool interactive, robust and scalable. A web tier application framework consists of three components: client tier, middle tier and information tier.

The client tier models the interaction with the user. It communicates with the middle tier via standard protocols and sends and receives standard data formats that meet user's needs. Clients supported by RSPL range from devices running standard Web browsers to pervasive devices such as PDAs.

The middle tier includes standard-based web servers for interacting with the client tier and executing business logic functions. It collects and assembles web pages composed from static to dynamic contents and delivers them to the clients. The Information tier is the data store for all application artifacts such as existing and new internal applications, services and data.

MVC has been recommended as the architectural design pattern for interactive applications that provide a host of design benefits such as

separating design concerns from content presentation of the web-tier application framework [19]. MVC separates design concerns from presentation content using three parts namely the Model logic responsible for handling business logic and functionality of the application in the middle tier. The MVC View is designated for content presentation to different client types. The Controller logic processes user inputs and requests by communicating between the information and client tiers. Table 3 illustrates how RSPL maps to the features of MVC and web tier framework.

RSPL	MVC	Web tier
Domain Knowledge	Model	Information tier
User Perspective	View	Client
Requirements Generator	Controller	Middle tier

Table 3: Mapping MVC features to RSPL

The RSPL Domain Knowledge component is mainly responsible for storage and documentation of domain artifacts that resulted from elicitation, scoping and specification of requirements as well as business functionalities from the domain.

With respect to MVC and the web tier application, the Domain Knowledge part maps to the Information tier and the Model component of MVC as they both signify the business logic, or application functionality and data storage. The RSPL User Perspective component is equated with the View portion of MVC as they both represent the presentation logic that provide

the interaction interface between an actor and the tool. Actors make their contributions to RSD construction (elicit, scope, and tailor requirements) through this component.

The RSPL Requirements Generator processes actor input and requests and, therefore, maps to the Controller part of the MVC pattern. In MVC, the Controller communicates with the Model to process View requests, while in RSPL, the Requirement Generator communicates with the domain Knowledge to turn actor-perspective into concrete requirements specifications.

CHAPTER FOUR

4.0 RSPL TOOL DESIGN AND IMPLEMENTATION

4.1 Introduction

This chapter provides a typical illustration of how the presented model could become the heart of an interactive, web based tool - RSPL. Initially the model is analyzed to identify and determine tool requirements. The analysis uses object-oriented techniques to model the tool requirements. The design is then presented to depict how the tool is to be implemented on a web tier application framework.

4.2 RSPL Tool Decomposition

The purpose of this section is to obtain requirements that necessitate the development of the tool. A system development life cycle approach supported by object-oriented analysis techniques was adopted to facilitate the identification of requirements. The system development cycle consists of five main phases namely Requirements Gathering, Requirements Analysis, Design, Implementation and Testing. These phases are described in detail in the subsections that follow.

4.2.1 Tool Requirements Definition Phase

RSPL identifies three actors who interact with the tool. The actors include the Domain Expert responsible for elicitation, the System Analyst responsible for scoping and the Developer responsible for requirements specification. Based on this preliminary requirements gathering, the tool implements the following key functionalities (see figure 11):

i. Logon Management: for identifying the actor currently using the tool and maintaining their profiles;

ii. Requirements Elicitation: for eliciting domain information from business operating environment;

iii. Domain scoping: to identify commonality and variable features from elicited and analyzed information;

iv. Requirements specification: for tailoring and customizing requirements or features to suit actor perspective;

v. Requirements generation and

vi. Administration: to allow actors access the information they need, e.g., search, suggested content, links, etc.

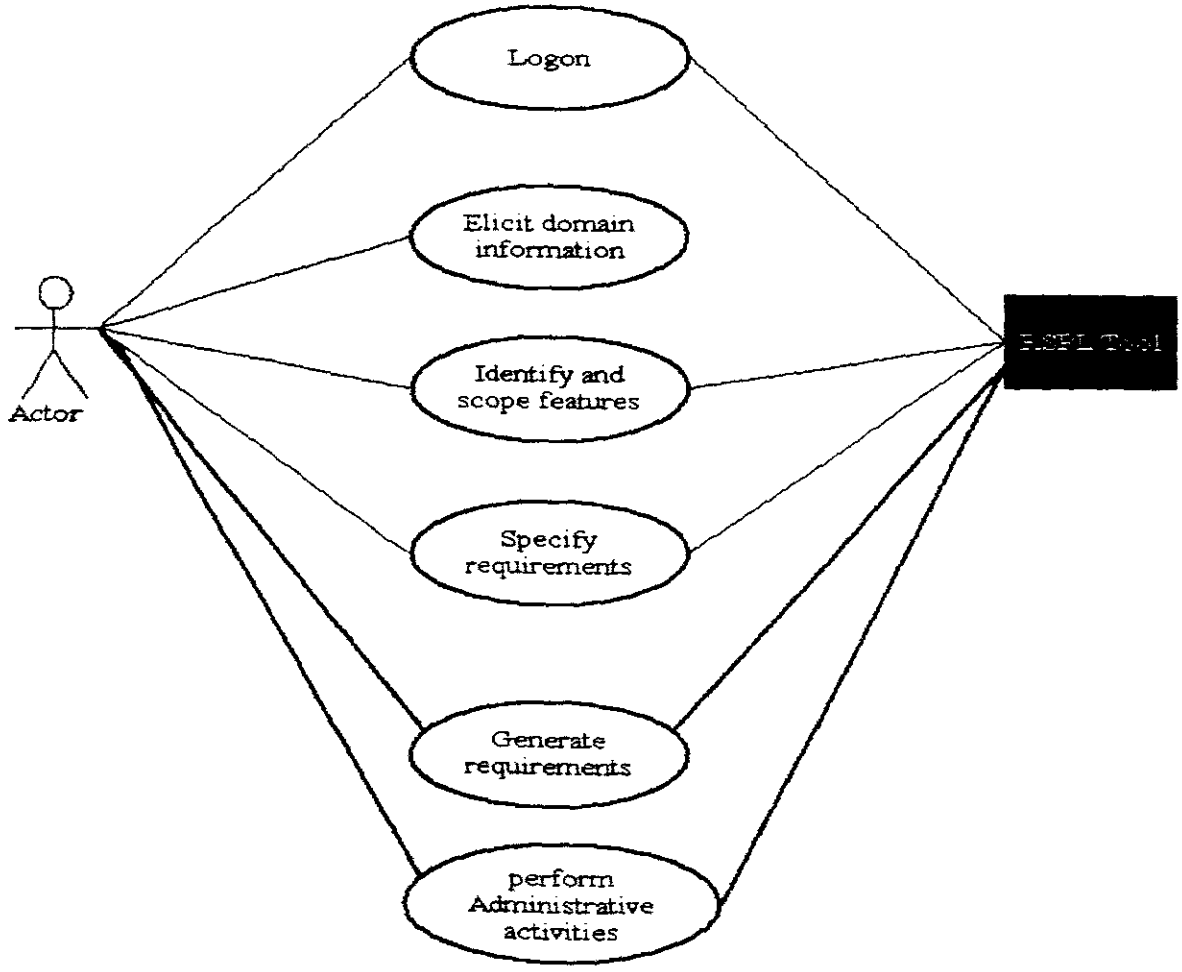


Figure 11: RSPL Use case

4.2.2 RSPL Requirements Analysis

4.2.2.1 Logon

The Logon use case is responsible for ensuring that security protocols are observed such as confirming that the right actor is using the system. Each actor is assigned administrative rights depending on the role being played i.e. domain expert, analyst or developer. For example, a domain expert whose role is eliciting information from the domain environment may not be presented with a requirement specification web page but with a domain elicitation and scoping page. Administrative rights require that an actor logs onto the system; and provides his particulars, such as level of expertise (job title/actor role), username and password. User name and password are checked for authentication purposes. Once authenticated, an appropriate message is sent back to the actor stating as either a success or a failure in the login status. If access is granted, the tool directs the actor to the respective web page that performs that functionality or transaction. Alternatively if the logon failed the actor is requested to try once again.

Figure 12 shows a sequence diagram of a logon use case. The flow of time is shown from top to bottom, that is, messages higher on the diagram happen before those lower down. The horizontal boxes are instances of the represented classes, and the vertical bars below are timelines. The arrows (links) are messages - operation calls and returns

from operations. The hide and show messages use guards to determine which to call. Constraints on the message are presented using square brackets [ ] and a message is sent only if the constraint is satisfied.

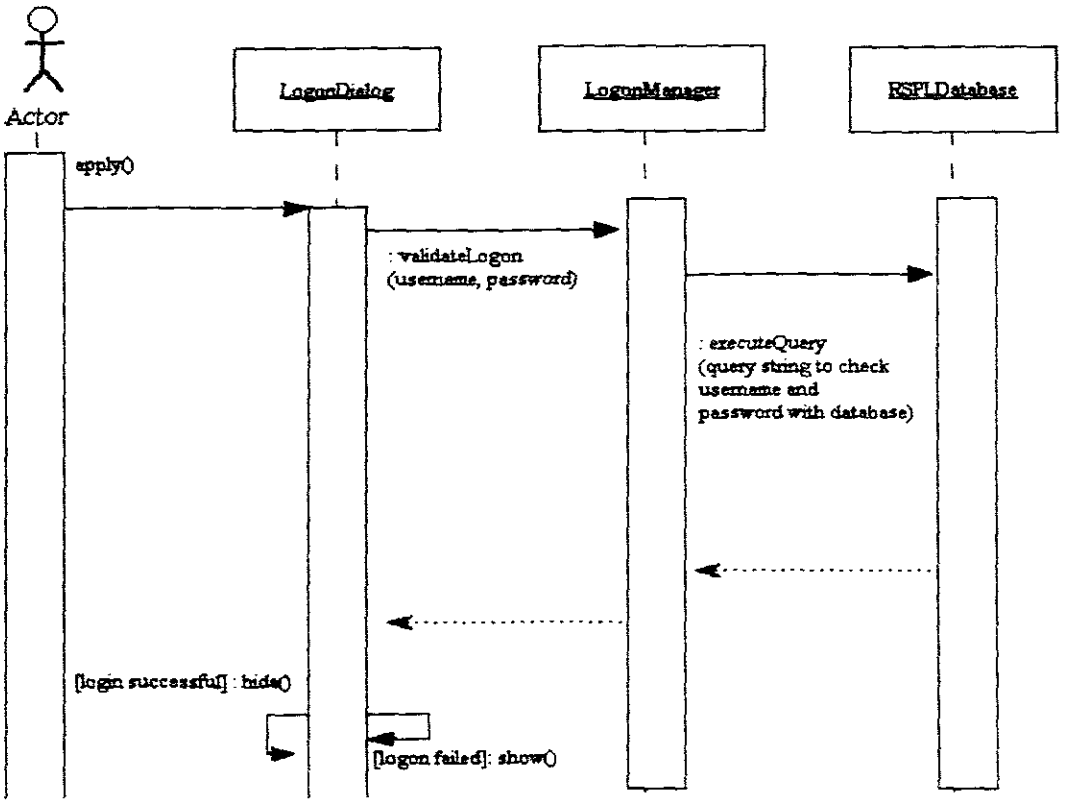


Figure 12: A sequence diagram for the login use case

The messages are labeled with the operation being called and parameters are shown. An RSPLDatabase class allows the queries to be executed against the database. When a query string is sent the ResultSet of the data is returned. The administrator administers the portal to ensure security issues are adhered to. He is to guarantee that only specific users are able to access the tool and use specific web pages. A single sign on approach is adopted to allow users to log on and

have access to the tool once. The tool then passes their authentication sites to other sites so that no logging in again is required.

4.2.2.2 *Requirements Elicitation and Scoping*

Requirements elicitation and scoping are interwoven processes, meaning that they depend on each other. Requirements Scoping depends on elicited information. Elicitation is a process in which actors, domain experts in particular, identify and document new or improved domain features for the purposes of acquiring sufficient and comprehensive domain knowledge about the problem domain. Domain scoping aims at selecting, based on the results of the elicited information, the systems that are included in the system-family and, secondly, what features are shared in the system-family [57]. It helps to describe the problem domain features, that is, system functions, interfaces, business rules, forms and reports, system performance factors, and quality attributes. Because the two processes are dependent on each other, a single sequence diagram was constructed in figure 13.

The flow of time is shown from top to bottom, meaning that elicitation takes precedence followed by scoping. An actor triggers the tool when he/she requests to define or elicit features. When an actor submits defined features during elicitation, the tool requests the actor to confirm the defined features. Once confirmed, the features are documented and

added as new domain information. During scoping of the domain, elicited information is retrieved and analyzed to identify reusable features. Once identified, features are checked by the FeatureManager as to whether they conform to business integrity rules before documentation as domain knowledge in the RSPLDatabase.

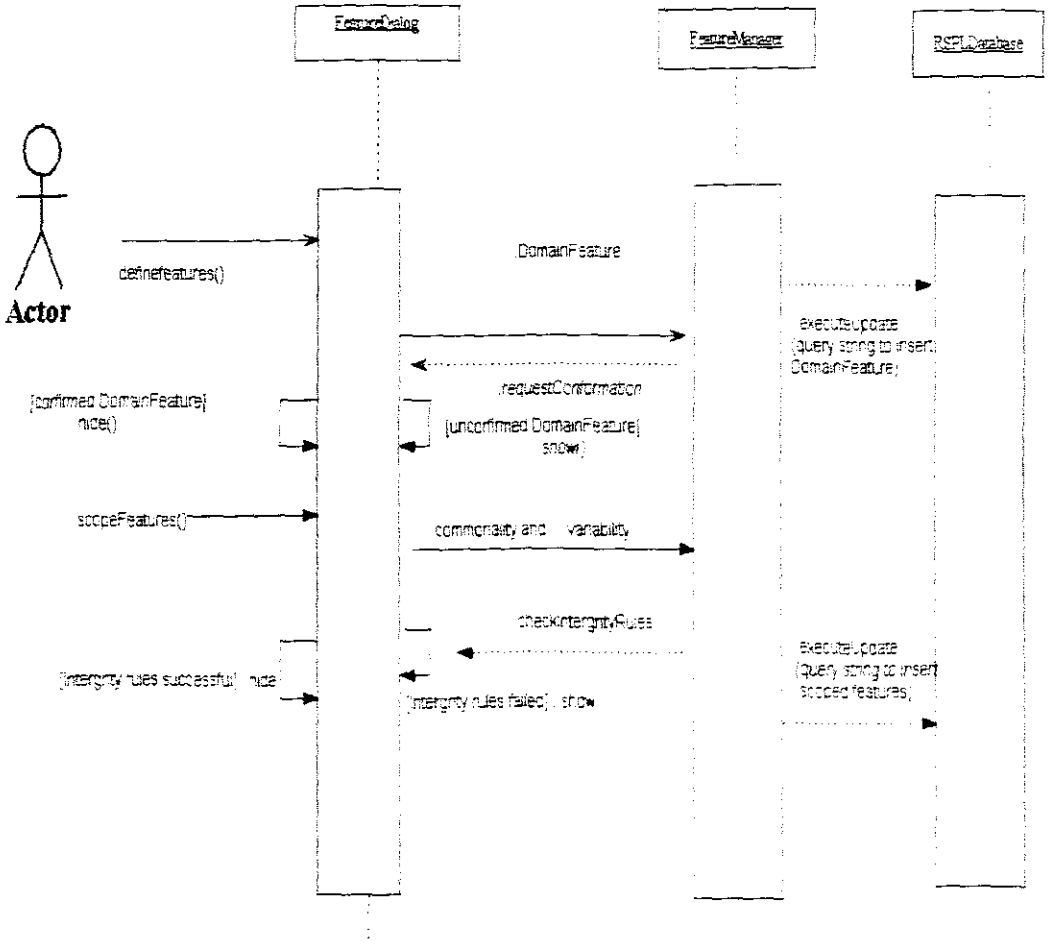


Figure 13: Elicitation and Scoping Sequence Diagram

4.2.2.3 *Requirements Specification Use Case*

Requirements Specification is an actor as well as template-driven process. An actor triggers the specification process once he/she

requests to tailor the domain features to generate specific requirements. The tailoring of features takes place using the FeatureDialog which then communicates to the FeatureManager, requesting actor specific requirements. The FeatureManager retrieves the feature datasets from RSPLDatabase, and via FeatureDialog, presents them to the actor who makes a choice of either to select one of them or to create/add new features. Figure 14 illustrates a Requirements Specification Sequence diagram. RSPL will then request the confirmation of selected features. If confirmed, RSPL slots it into the appropriate RSD template section otherwise those features are removed.

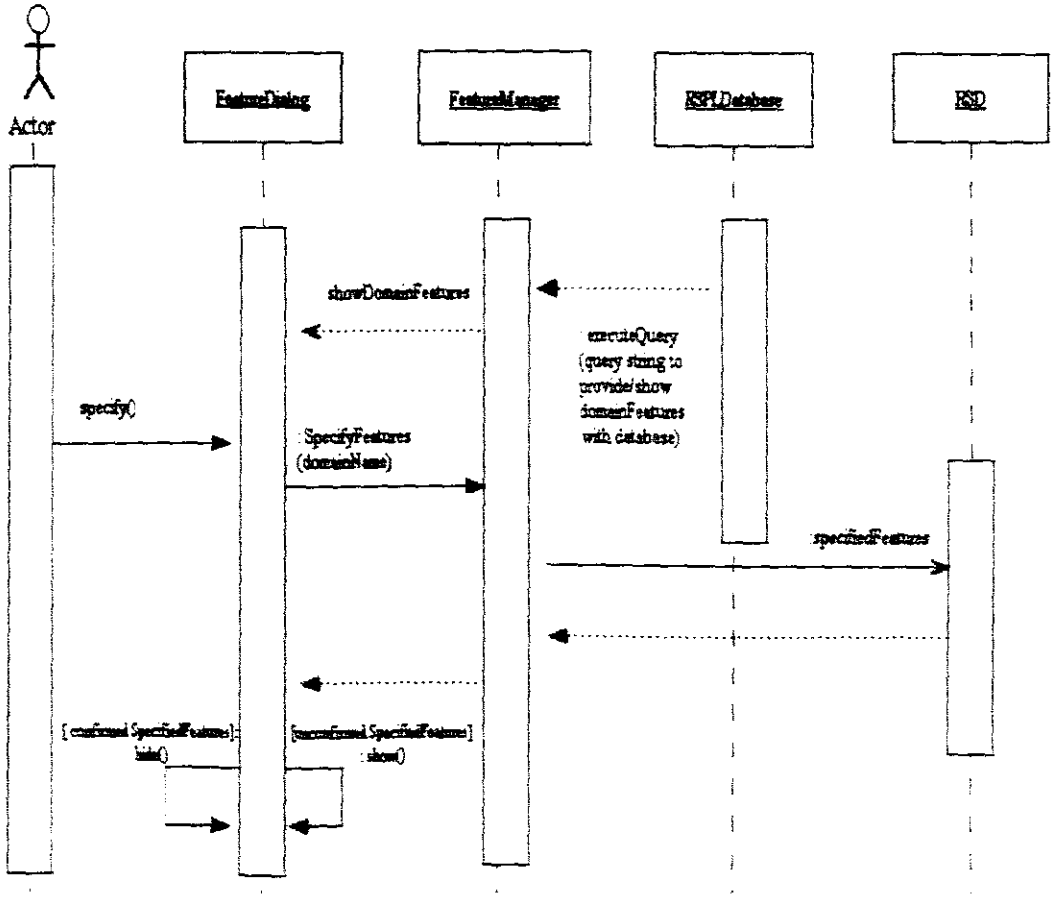


Figure 14: Requirements Specification Sequence diagram

4.2.3 Tool Design

The six use cases were repackaged into four tool capabilities that must be provided in order to accomplish the RSPL goals. These capabilities are shown in figure 15 as: Domain Browsing, Editoring, Requirement Generation, and Knowledge Base.

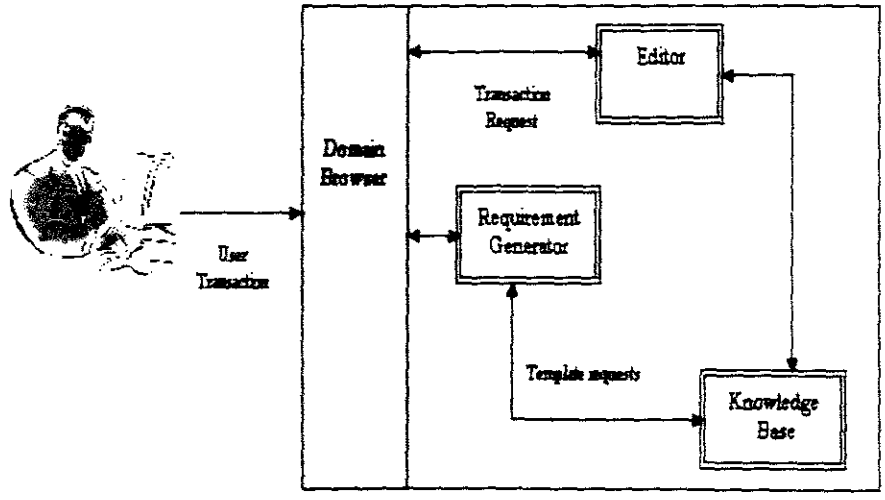


Figure 15: RSPL Tool Architecture

4.2.3.1 Browsing

The Domain Browser provides the interactive user interface between an actor and the tool, making it possible for them to make requests such as viewing and requesting information from the knowledge base. Actors view the application output in the browser and click hyperlinks and form buttons to interact with the application. When an actor makes a request, the

browser communicates with the Requirement Generator to retrieve data from the Knowledge Base, i.e. Domain Knowledge.

Each web page (servlet) employs an XSL transformer to generate appropriate presentations for each client type. Each client type requires that the application have a separate set of XSL transformations for it to deliver the content.

4.2.3.2 Editing

The Editing capability provides the necessary software support for tasks such as elicitation, scoping and tailoring of domain information to generate specific requirements according to the respective actor's perspective and preferences. The Editing capability supports the following operations:

i. Addition of new features: creation of either a new product instance, or simply incorporation of additional features;

ii. Deletion of a feature: removing that feature from the domain;

iii. Updating a feature: making changes to current or existing features to ensure that they have current or up-to-date information and

iv. Searching: finding specific features in the domain knowledge.

4.2.3.3 *Requirement Generation (RG)*

The Requirement Generation process requires that the content of the knowledge base is transformed into the view presented to the user by the domain browser. The process consists among other steps of querying the Knowledge Base and retrieving requested specified requirements. The web tier application framework enables RG to have a number of web components and extensions such as Enterprise JavaBeans (EJB); Java Database Connectivity for accessing database services; and support for creating, parsing and translating XML documents.

Figure 16 provides a detailed architecture of RSPL, illustrating how it relates to the web tier-application framework and MVC. Further explanation is provided by Deitel et al in [58]. One of the supported web components are Java servlets to provide most of the user interactivity features. The Domain Knowledge component and Requirement Generator are supported by the enterprise JavaBeans (EJB) components provided by web applications server. The figure shows how different clients communicate with the web server which forwards the requests to the servlets running in the application server's servlet container. Each client's request is transformed using XML Stylesheet Language (XSL) implemented at the presentation logic. The XSL processor allows the actor to take the abstract semantics of an XML document and transform it into a presentation language suitable for the client type (for example WAP

and i-Model clients). MVC makes this possible by adapting the new client type to operate as an MVC view.

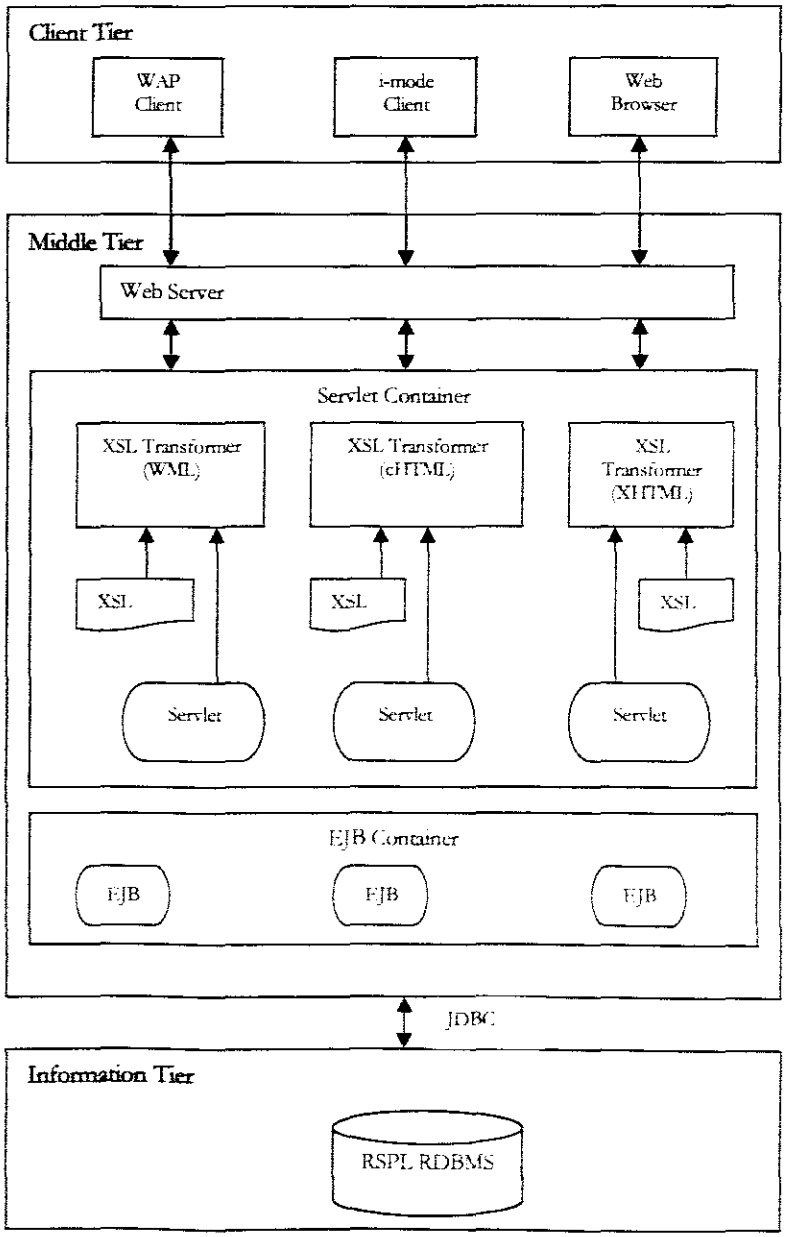


Figure 16: Detailed MVC architecture of RSPL Tool

RG is a stateful session EJB that represents an actor's specifications, stored in an RSD template. The RSD stateful session EJB manages the actor's specifications and is the primary RG component in the RSPL tool.

This is because sometimes actors browse through the tool and add features to their RSD, only to discover later on that those features do not meet their requirements. Rather than storing such an RSD, the EJB container removes the RSD EJB instance from RG. RG also implements the application's presentation logic, making it possible to present content to different client types.

4.2.3.4 Knowledge Base (KB)

The Knowledge Base represents the information tier logic, responsible for data maintenance of elicited and scoped information as well as generated domain artifacts. KB supplies the feature and function of an object-relational database to RSPL applications that include querying and updating access to database information through EJB using Structured Query Language, JAVA SQL and JDBC interfaces.

KB provides logical database description, i.e. the schema, used to describe and specify the artifacts and the relationships among them. Artifacts in this aspect include entities or classes, attributes and their relationships. It also provides a specific view of the data item types and record types to be used by an actor. The KB consists of a catalogue of all data types in the Knowledge Base that is used to document and manage domain vocabulary. A domain vocabulary gives the data type's names, definitions and characteristics of each domain artifacts. The adopted web-tier application framework design enables RSPL to separate design

concerns from its presentation and control logic by having three packages: Servlet, Model and EJB. The different MVC components i.e. Model, View and Controller are shown on the horizontal top row. The RSPL packages are indicated vertically from top to bottom on the first column in Table 4 shows how each RSPL package maps to an MVC component.

Package/MVC	Model	View	Interface	Controller
Servlet		XSL: Xalan and Xerces XSL Transformer		XML Servlet
				Register and Login
				Add/Remove and Update
				Search/View
				RSD (Requirement Specification Document)
Model	Domain			
	Product			
	Feature			
	Service			
	Actor			
EJB	RSD EJB		RSD Remote and RSD Home	
	Domain EJB		Domain Remote and Home	
	Product EJB		Product Remote and Home	
	Feature EJB		Feature Remote and Home	
	Service EJB		Service Remote and Home	
	Actor EJB		Actor Remote and Home	

Table 4: RSPL tool packages

CHAPTER FIVE

5.0 RSPL TOOL CASE STUDY

5.1 Introduction

The goal of this chapter is to demonstrate how the RSPL tool was implemented by transforming artifacts from analysis and design into an *executable system. The content is a follow up to design and* implementation reported in chapter four. An e-Commerce domain has been selected to demonstrate the working of the tool.

5.2 Case Study: Generation and Specification of requirements for the E-Commerce Domain

5.2.1 Rationale for an E-Commerce Domain

E-Commerce is the means of selling goods and services on the Internet in a digital format. The increasing demand of E-Commerce Customers' expectations requires software engineers to have tool support that will help in the provision of timely information *and services.* Wierling [60] observed that *E-commerce development environment involves shorter release cycles, higher quality releases, and ever-changing end user demands.* To this effect, the tool support becomes a necessity in meeting customers changing expectations. This case study is presented as an online web store consisting of two main product categories namely, the

ordinary *online web store* and the *mobile web store*. Each product category is differentiated into three different instances depending on the number or level of features. The instances include:

i. *Mini Web Store (MWS) features:* These are necessary features that are required for a web store to be functional. MWS online web store will have two minimal features: Catalogue Management which includes controlling and displaying product range; and an Ordering System responsible for placing and processing customer orders.

ii. *Standard Web Store (SWS) features:* These are average features that include minimal web store features and additional features that offer differentiated characteristics. SWS online web store category will implement Catalogue Management; eSales Management which involves orders and billing management, reports generation and analysis, marketing and advertisement though banners; and Procurement which includes handling inventory and integrating transaction between buyers and suppliers either by event notification such as SMS or email.

iii. *Deluxe Web Store (DWS) features:* These are specialized features that include both mini and standard web store features. Compulsory features implemented will include Catalogue Management; eSales Management; Procurement; Customer Relationship Management, ensuring provision of customer self services and direct marketing using PDAs for subscribed customers; and Website Management to handle *content layout, web load testing and functional testing.*

This case study looks at the possibility of RSPL generating specific requirements for the SDLC actors who intend to provide Online Web Store services for the tourism industry sector.

5.2.2 The Portal Interface

The Domain Browser provides an interactive interface between the actor and the tool. Figure 17 presents a prototype of an RSPL portal interface. Links from various tool sources (other web pages) have been consolidated to provide a single portal entry access to information. To ensure that the tool is easier to use, actors are provided with a walkthrough link about the tool and a tutorial that provides them necessary assistance with using tool functionalities.

Figure 17: RSPL Portal Interface

5.2.3 User Authentication

User authentication is a process of confirming each actor's information before using the tool. An actor is required to log onto the system and provide his/her particulars such as level of expertise (job title/actor role), username and password (UserID). The system then checks whether the actor exists within RSPLDatabase. Figure 18 shows the scenario of a logon session.

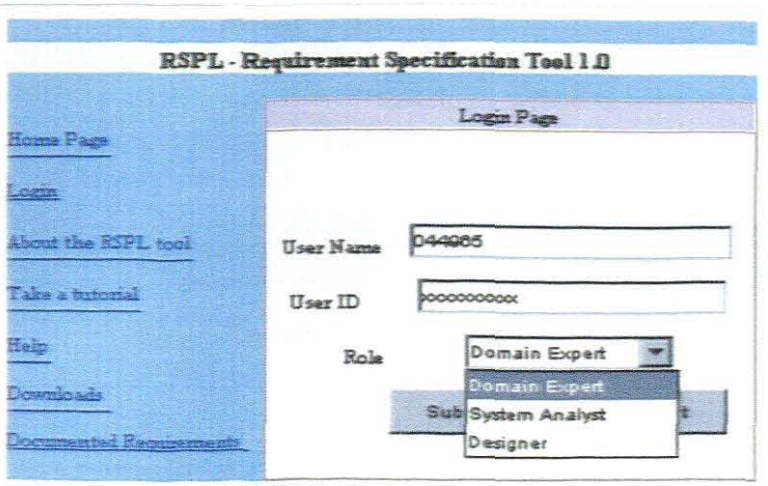


Figure 18: RSPL Login Page

5.2.4 Elicitation and Scoping Interface

Requirements elicitation is carried out by the domain expert who has full understanding of the domain in terms of the stakeholder needs and possible environmental changes. Requirement templates for eliciting and scoping are structured to capture domain features such as domain name, its description, goal and actors. A domain template snapshot that elicits elementary domain features is presented in figure 19. While figure 20 elicits and scopes behavioral features, figure 21 elicits quality features.

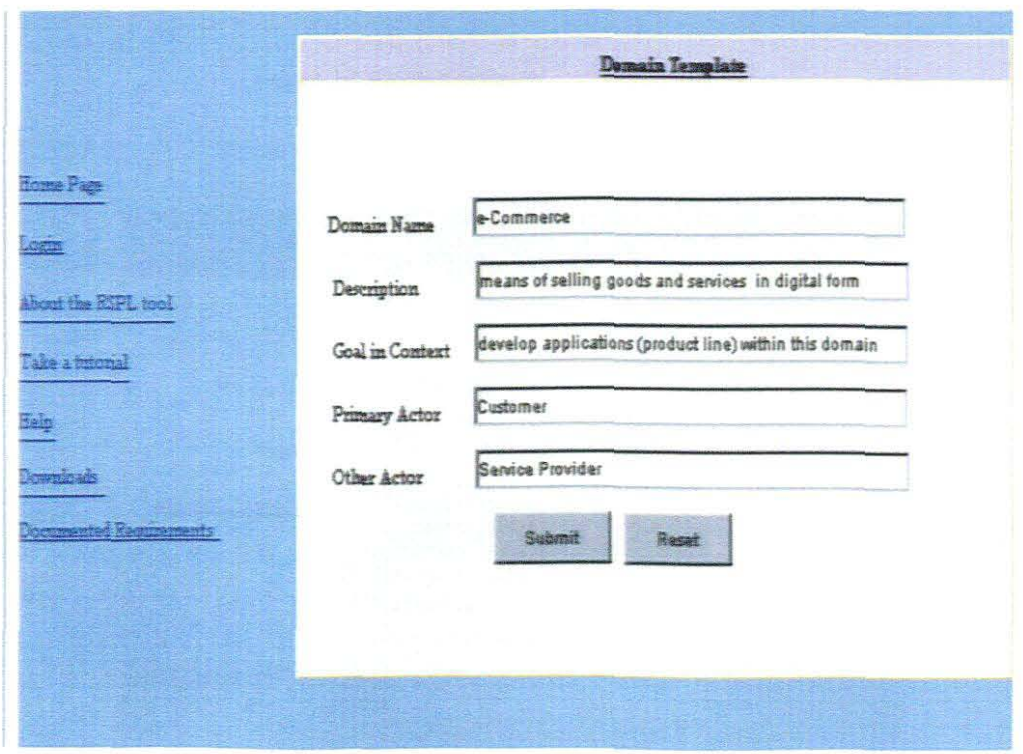


Figure 19: Domain Template Elicitation

5.2.5 Specification and Generation Interface

The model that drives this process is domain knowledge derived from various domain sources such as actor perspectives, domain vocabulary and knowledge base. Each source is a dynamic content repository corresponding to a web mini application on a portal server.

The requirements specification process requires that an actor selects features from predefined domain knowledge. Figure 22 shows an actor specifying the domain types, which in this case study, is the e-Commerce domain. The different product instances that characterize the domain are displayed in figure 23 and the actor makes a choice of deluxe web store that offers specialized services.

Figure 22: Domain Template Specification

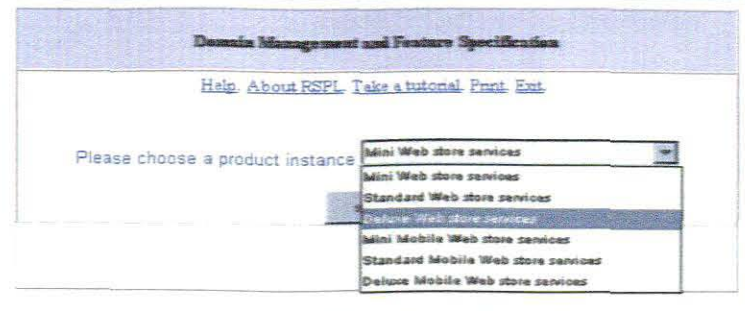


Figure 23: Selection of a product Instance

Automatic generation of a deluxe Online Web store features are reflected in figure 24. The actor is then allowed to specify product features as per product requirements. Any selected feature can have sub features that also require specification, for example, figure 25 shows sub features of figure 24 of the selected feature. If no further sub features are found, an RSD is then automatically generated. In this case, a choice of Layout Maintenance is selected although further choices can also be made. The Layout Maintenance feature has no further sub features to be specified. This allows the automatic generation of abstract semantics in figure 26 and a partial RSD in figure 27.

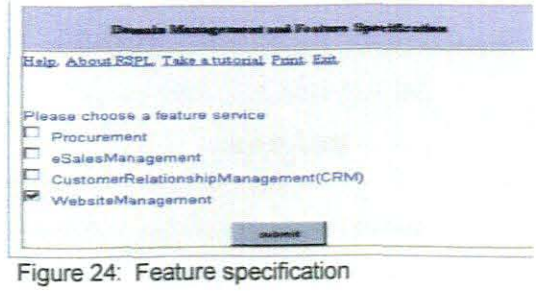


Figure 24: Feature specification

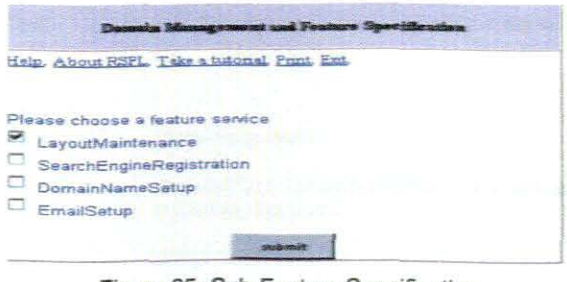


Figure 25: Sub Feature Specification

Figure 26: A partial metadata representation of the Entity Domain.

REQUIREMENTS SPECIFICATION DOCUMENT TEMPLATE

Help AboutRSPL Take a Tutorial Print Exit

INTRODUCTION

Domain Name:	e-Commerce Application
Domain Description:	Selling goods and services in digital format
Goal:	Develop application instances within this domain
Constraint:	Difficult in obtaining all stakeholder information as some may not have internet connection

Domain Features:

Product Instance:	Deluxe Webstore Services
Description:	Specialized features that include both mini and standard features as well as compulsory features.
Features:	Web site Management CustomerRelationshipManagement(CRM) eSalesManagement Procurement

FUNCTIONALITIES

Web Site Management Functionalities:

Manage Session
Manage Component

Actors SiteManager

Figure 27: A partial RSD for the Online Ecommerce Application

5.3 Evaluation of Result

The development of the RSPL tool using a web-tier application framework and a model-driven architecture made it possible to implement a tool with the following functionalities or capabilities:

i. Tool support for requirements engineering tasks such as elicitation, scoping and specification;

ii. Generation of requirements metadata during product requirements specification phase such that its equivalent metadata is automatically generated without the actor's efforts and

iii. Support for mobile client types using J2EE platform.

CHAPTER SIX

6.0 CONCLUSION AND FUTURE WORK

6.1 Conclusion

As software development moves from a single product development to a family of systems, requirements engineers need to rethink new models for tool support. The tool support is an essential part in software development as it (i) decreases time to market by automatically generating requirements or code (ii) decrease errors due to limited human intervention (iii) increases return on investment by decreasing overhead costs. The Model Driven Architecture (MDA) benefits seem to provide solutions to software engineers RE problems. Although the MDA benefits have featured in application design and implementation phase of system development, this work extends the MDA benefits to the requirements phase as a solution to tool support for product line development. This research work addresses tool support concerns by proposing a requirements engineering model (RSPL) for product line software engineering.

The first objective was to construct a model to support automatic transformation of domain features into actor-specific requirements. Following the Model Driven Architecture principles, a Requirement

Specification Model for a product line (RSPL) based on three constructs (i) Domain Knowledge (ii) User Perspective and (iii) Requirement Generation was formulated.

It has been shown that the Domain Knowledge component is a centralized repository that does not reference any particular system implementation or technology, but is responsible for capturing and documenting the information viewpoint of a target application or a conceptualization of the application. The study further shows how through the User Perspective component, actors are able to tailor domain knowledge to suit requirements for a specific application instance using two main templates (i) domain template and (ii) product template. The tailoring of requirements entail transforming the generic domain knowledge into specific requirements through a portal-type interface. Requirements are then generated using standardized templates through the Requirements Generator component. The requirement generation process is dictated by actor specific requirements, specified using both domain and product templates. The generated requirements are then presented to the actor in a requirements specification document.

The Second objective was to provide a reusable implementation of the model for different client types. This was achieved by adapting a web tier application framework which provides interactive applications, multi-user capability (support for different client types) and provision of a host of

design benefits. The implementation is achieved using an e-Commerce application domain case study which demonstrates how the RSPL tool was implemented. Moreover, the tool offers guidance and active support to novice users and those from different background, through tutorial and proposing various kinds of links automatically.

The RSPL tool is not a full feature tool like Holmes Tool [61] but it serves the purpose for which it is designed. The underlying model proves that a model-driven software requirements factory is a future possibility. A true factory will allow actors to play specialized roles in the production line as RSPL has demonstrated while delivering an automatically generated finished product.

The tool does not have a validation capability but this can be plugged into a production version. It however demonstrates that both requirements artifacts and design artifacts can be stored side by side in the knowledge base for collaborative update. Any part of software documentation can then be generated just as in the case of requirements.

6.2 Future Work

The RSPL model was not designed with specific security, scalability and performance quality of service in mind. For it to be used in a production environment, design criteria for the Quality of Services issues must be evaluated first. The model does not also provide validation capabilities essential for checking requirements completeness and consistency. A

decision model that flags requirements inconsistency and dependency constraints based on the approach by Feather in [59] can be adopted before introducing RSPL to the production environment. These aspects of the research are deferred to the future.

Another weakness of current RSPL tool implementation is that while MDA and MVC architectures provided opportunity for reuse, this would have been enhanced by using portlet technology to structure the portal implementation. With the advent of portal technology API and SDK, all future versions of this tool will take advantage of the extra modularity, orthogonality and maintainability that portlets present.

RSPL has been designed with multimodality in mind, however, the model should be enriched further in future to enable concrete extensions for mobile clients either micro-browser or operating environments.

REFERENCES

[1] Bosch, J. (2000). *Design and Use of Software architectures: Adopting and evolving a product line approach*. Addison Wesley Professional.

[2] Bowen, B. J. (1978). *Are current approaches sufficient for measuring software quality?* ACM SIGMETRICS Performance Evaluation Review, vol.7, Issue 3-4, pp. 148 – 155.

[3] Brown, A. (2004). *An introduction to Model Driven Architecture Part I: MDA and today's systems*. The Rational Edge: Copyright IBM Corporation. Available at http://www-128.ibm.com/developerworks/rational/library/content/RationalEdge/feb04/3100.pdf Date last accessed November 2004.

[4] Bryant, B. and Pan, A. (1991). *Formal Specification of Software Systems using Two-Level-Gramma*. COMPASAC'91 15th Annual Intl. Computer Software and Applications Conf. pp155-160.

[5] Cockburn, A. (2001). *Writing Effective Use Cases*. The Crystal Collection for Software Professionals. Addison Wesley.

[6] Cohen, S. and Northrop, L. (1998). *Object-Oriented Technology and Domain Analysis*. Fifth International Conference on Software Reuse (ICSR'98), pp86.

[7] Cox, K. (2000). *Fitting Scenarios to the Requirements Process.* IEEE Proceedings of the 11th International Workshop on Database and Expert Systems Applications (DEXA'00), pp. 995.

[8] DeBaud, J. (2000). TrueScope Technologies Inc, Software Product Families Solutions.

[9] Deelstra, S. Sinnema, M. Van Gurp J. and Bosch, J. (2003). *Model Driven Architecture as Approach to Manage Variability in Software Product Families.* Proceedings of the Workshop on Model Driven Architecture: Foundations and Applications (MDAFA 2003). CTIT Technical Report TR-CTIT-03-27, University of Twente, pp.109-114.

[10] Dionisi Vici, A. Argentieri, A. Mansour, A. d'Alessandro, M. Favaro, J. (1998). *FODAcom: An Experience with Domain Modeling in the Italian Telecom Industry*, *Proceedings of IEEE ICSR5*, pp 166.

[11] Du Bois, P. (1995). *The Albert II Language – On the Design and the Use of a Formal Specification Language for Requirements Analysis.* PhD. thesis, Dept. of Computer Science, University of Namur, Namur, Belgium.

[12] Firesmith, D. (2003). *Modern Requirements Specification. Journal of Object Technology*, Vol.2, No.2, March-April 2003, pp.53-64.

[13] Flurry, G. and Vicknair, W. (2001). *The IBM Application Framework for e- business.* IBM Systems Journal, Vol.40, No.1 pp 8. Available at http://www.research.ibm.com/journal/sj/401/flurry.html. Date last accessed August 2005.

[14] Frincke, D. Wolber, D. Fisher, G. and Cohen, G. (1992). Requirements Specification Language (RSL) and Supporting Tools. Available at http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19930003157_1993003157.pdf Date last accessed June 2004.

[15] Gomaa, H. Kerschberg, L and Farrukh, G.A. (2000). *Domain Modeling of Software Process Models.* Sixth IEEE International Conference on Complex Computer Systems (ICECCS'00), pp50.

[16] Gomaa, H. and Kerschberg, L. (1995). *Domain Modeling for Software Reuse and Evolution.* Proc. IEEE International CASE Conference, Rio de Janeiro, Brazil.

[17] Grunbacher, P. and Braunsberger, P. (2003). *Tool Support for Distributed Requirements Negotiation.* In Cooperative methods and tools for distributed software processes, Cimititle, A., De Lucia, A., and Gall, H., Editors. FrancoAngeli: Milano, Italy. pp 56-66.

[18] Hepper, S and Hesmer, S, (2005). *Introducing the Portlet Specification, Part 1.* JavaWorld.com, an IDG company. Retrieved from http://www.javaworld.com/javaworld/jw-08-2003/jw-0801-portlet.html. Date last accessed May 2004.

[19] Herlea, D.E. (1998). *Users involvement in requirements engineering.* In the Proceedings of the Workshop on Internet-based groupware for users involvement in software development, Seattle, USA. Available at http://ksi.cpsc.ucalgary.ca/KAW/KAW96/herlea/FINAL.html Date last accessed September 2004.

[20] IEEE, (1993). Standard VHDL Language Reference Manual. Standard 1073-1993, New York.

[21] Jarzabek, S. Chun, O. W and Zhang, H. (2003). *Handling Variant Requirements in Domain Modeling.* Journal of Systems and Software. Vol.68, No. 3, pp 171 - 182 .

[22] John, I. and Dörret, J. (2003). *Extracting Product Line Model Elements from User Documentation.* Technical Report, Fraunhofer IESE. Available at http://www.iese.fhg.de/pdf_files/iese-112_03.pdf Date last accessed July 2004.

[23] Kan, S. H. (2003). Metrics and Models in Software Quality Engineering. 2nd Edition. Addison Wesley Professional.

[24] Kang, K. C. Lee, J and Donohoe, P. (2002). *Feature-Oriented Product Line Engineering*. IEEE Software 19, (4) (July/August 2002), pp 58-65.

[25] Kang, K. Cohen, S. Hess, J. Novak, W. and Peterson, A. (1990). Feature-Oriented Domain Analysis Technical Report No. CMU/SEI-90-TR-21, Software Engineering Institute.

[25] Kassel, W.N. and Malloy, B.A. (2003). *An Approach to Automate Requirements Elicitation and Specification.* Proceedings of the 7th IASTED International Conference Software Engineering and Applications, pp 397-029. Available at http://www.cs.clemson.edu/~malloy/papers/sea03/iasted.pdf Date last accessed September 2004.

[26] Griss, M. Favaro, J. and D'Alessandro, M. (1998). *Integrating Feature Modeling with the RSEB.* Proceedings of Fifth International Conference on Software Reuse. Available at http://www.favaro.net/john/home/publications/rseb.pdf Date last accessed November 2004.

[27] Maiden, N.A.M. ARTSCENE Scenario Presenter. Available at www.soi.city.ac.uk/artscene. Date last accessed November 2005.

[28] Meier, J.D. Vasireddy, S. Babbar, A. and Mackman, A. *(2004). Chapter 1 — Fundamentals of Engineering for Performance: Improving .NET Application Performance and Scalability.* Available at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/scalenetchapt01.asp. Date last accessed June 2005.

[29] Meservy, T.O. and Fenstermacher, K.D. (2005). *Transforming Software Development: An MDA Road Map.* IEEE Computer Society.

[30] Metzner C. Cortez L. and Chacín D. (2005). *Using Blackboard Architecture in a web application.* Issues in Informing Science and Information Technology, pp 743 - 755. Available at http://2005papers.iisit.org/I58f73Metz.pdf Date last accessed April 2005.

[31] Moon, M and Heung, S.C. (2005). *An Approach to Developing Domain Requirements as a Core Asset Based on Commonality and Variability Analysis in a Product Line.* IEEE Transactions on Software Engineering, July 2005 Vol. 31, No.7. pp 551 – 569.

[32] Object Management Group. (1999). *OMG Unifies Modeling Language Specification UML v1.3.* Technical Report, Document ad/99-06-08, Object Management Group (OMG).

[33] Padmanabhan, P. (2001). *DECIMAL: A Requirements Engineering Tool for Product Families.* Oiwa State University. Ames, Iowa. Available at http://archives.cs.iastate.edu/documents/disk0/00/00/02/79/00000279-00/thesis.pdf Date last accessed September 2004.

[34] Parnas, D.L. (1976). *On the design and development of program of families.* IEEE Transactions on Software Engineering, vol.2, no. 2, pp 1-9.

[35] Regnell, B. Kimbler, K. WessXn A. (1995). *Improving the Use Case Driven Approach to Requirements Engineering.* In Proceedings of the Second IEEE International Symposium on Requirements Engineering, pp 40-47.

[36] Smith, J.R.W and Reed, R. (1989). *Telecommunications Systems Engineering using SDL.* Amsterdam. The Netherlands: NorthHolland /Elsevier.

[37] Schmid, K. (2002). *A Comprehensive Product Line Scoping Approach and Its Validation.* International Conference on Software Engineering, Proceedings of the 24th International Conference on Software Engineering., pp. 593-602.

[38] Schmidt, K. Shank, M. (2000). *PuLSE-BEAT: A Decision Support Tool for Scoping Product Lines.* Third International Workshop on Software Architectures for Product Families, pp197-203.

[39] Seyff, N. Grunbacher, P. Maiden, N. Toscar, A. (2004). *Requirements Engineering Tools Go Mobile.* International Conference on Software Engineering. Proceedings of the 26th International Conference on Software Engineering, pp 713 – 714. Available at http://csdl2.computer.org/comp/proceedings/icse/2004/2163/00/21630713.pdf Date last accessed May 2005.

[40] Sidky, A.S. (2003). RGML: *A Specification Language that Supports the Characterization of Requirements Generation Processes.* Available at http://scholar.lib.vt.edu/theses/available/etd-07292003-112122/unrestricted/Sidky_Thesis.pdf. Date last accessed February 2005.

[41] Spafford, G. (2004). *Understanding 'Mean Time Between Failure.* JupiterWeb networks. Available at http://itmanagement.earthweb.com/columns/article.php/3354191. Date last accessed June 2005.

[42] Stento, R. *Persistent Data Architecture Approach.* Available at www.objectstore.com. Date last accessed March 2005.

[43] Sun Microsystems Inc. (2002). *Designing Enterprise Applications with the J2EETM Platform.* 2nd ed. Sun Developer Network. Available at http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/ Date last accessed February 2005.

[44] Teichroew, D. Hersey III, E.A. (1982). *PSL/PSA: a computer aided technique for structured documentation and analysis of information processing systems.* In Advanced System Development/Feasibility Techniques, Wiley, New York, pp315-329.

[45] Wieringa, R. and Ebert, C. 2004. *RE'03: Practical Requirements Engineering Solutions,* pp 16-18. IEEE Software Computer Society. Available at http://csdl2.computer.org/comp/mags/so/2004/02/s2016.pdf Date last accessed February 2004.

[46] Tomas, D and Hunt, A. (2004). Nurturing Requirements. Software, IEEE Volume 21, No.2, pp 13 – 15.

[47] Kang, K.C. Kim, S. Lee, J. Kim, K. Shin, E. Huh, M. (1998). *FORM: A feature-Oriented Reuse Method with Domain-Specific Reference Architecture.* Annals of Software Engineering, Volume 5, No. 1, pp. 143-168.

[48] Frakes, W.B and Kang, K. (2005). *Software Reuse Research: Status and Future.* IEEE Transactions on Software Engineering, Vol. 31, No. 7, pp 529-536. Available at http://selab.postech.ac.kr/publication/2005_TSE_SoftwareReuseResearch.pdf Date last accessed June 2005.

[49] Fowler, M. *The New Methodology.* Available at http://www.martinfowler.com/articles/newMethodology.html, 2005. Date last accessed April 2005.

[50] Wikipedia. *Agile software development.* Available at http://en.wikipedia.org/wiki/Agile_software_development. Date last accessed December 2004.

[51] Kontio, M. (2005). *Architectural manifesto: MDA for the enterprise: An architect's approach to more productive development.* Available at http://www-128.ibm.com/developerworks/library/wi-arch16/ Date last accessed August 2005.

[52] Letelier, P. (2002). *A Framework for Requirements Traceability in UML-based Projects*. In Proc. of 1st International Workshop on Traceability in Emerging Forms of Software Engineering. In conjunction with the 17th IEEE International Conference on Automated Software Engineering, pp. 32-41. U.K: Edinburgh.

[53] Sabat, N. J. Leite Prado, J and Cysneiros, L.M. (2000). *Non-Functional Requirements for Object-Oriented Modeling.* III Workshop de Engenharia de Requisitos, pp 109-125. Available at http://wer.inf.puc-rio.br/WERpapers/artigos/artigos_WER00/sabat_neto.pdf Date last accessed November 2005.

[54] John, I and Dorr, J. (2003). *Elicitation of Requirements from User Documentation*, Available at http://crinfo.univ-paris1.fr/REFSQ/03/papers/P01-John.pdf Date last accessed June 2004.

[55] Fantechi, A. Gnesi, S. John, I. Lami, G. and Dorr, J. (2002) *Elicitation of Use Cases for Product Lines.* 9th IEEE Conference and Workshops on Engineering of Computer-Based Systems. Available at http://fmt.isti.cnr.it/WEBPAPER/25-PFE-ucf.pdf Date last accessed March 2005.

[56] Bosch, J. (1999). *On the Design of System Family Architectures.* Available at http://www.serc.nl/lac/LAC-2001/lac-1999/docs/jan_bosch.pdf. Date last accessed June 2004.

[57] Deitel, H.M. Deitel, P.J. and Santry, S.E. (2002). *Advanced Java 2 Platform: How to Program.* Prentice Hall.

[58] Feather, M.S. (1998). *Rapid Application of Lightweight Format Methods for Consistency Analyses.* IEEE Transactions on Software Engineering, vol.24, no.11, pp 949 – 959.

[59] Wierling, A. (2001). Requirements for E-Business Success. Technology Builders, Inc. Available at http://www.techlinks.net/DesktopModules/Community%20Publishing/Rss.aspx?TabID=92&ModuleID=430&CategoryID=5 Data last accessed August 2004.

[60] Succi, G. Yip, J and Pedrycz, W. (2001). *Holmes: An Intelligent System to Support Software Product Line Development,* pp. 0829 Proceedings of the 23rd International Conference on Software Engineering. Canada: Toronto. Available at http://www.unibz.it/web4archiv/objects/pdf/cs_library/holmes-anintelligentsystemtosupportsoftwareproductlinedevelopment.pdf Last accessed September 2005.